VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

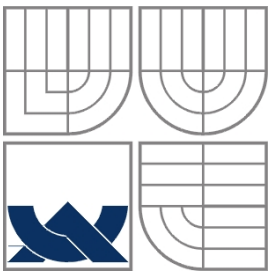
NÁSTROJ PRO ODPOSLECH OBSAHU KOMUNIKACE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

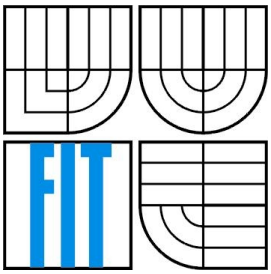
AUTOR PRÁCE
AUTHOR

ŠTEFAN ZIMA

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁSTROJ PRO ODPOSLECH OBSAHU KOMUNIKACE

CONTENT OF COMMUNICATION INTERCEPTION PROBE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ŠTEFAN ZIMA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. LIBOR POLČÁK

BRNO 2012

Abstrakt

Táto práce je zaměřena na tvorbu nástroje pro odposlech obsahu síťové komunikace. Rozebíra problematiku zákonných odposlechů a techniky urýchlení zpracování příchozího provozu v prostředí operačního systému Linux. Zaměřuje se na implementační techniky s využitím knihovny PF_RING. Aplikace je implementovaná v jazyce C a pak testována na komoditním hardvéru s využitim generátoru provozu.

Abstract

This thesis is focused on creation of tool for intercepting content of network communication. It discusses the legal issue of surveillance and techniques for acceleration of processing incoming traffic in the Linux operating system. The aim of this thesis are implementation techniques using PF_RING library. The application implementation in language C is then tested on commodity hardware using the traffic generator.

Klíčová slova

Zákonný odposlech, Zachytávaní paketů, PF_RING, Linux, libpcap

Keywords

Lawful interception, Packet capture, PF_RING, Linux, libpcap

Citace

Štefan Zima: Nástroj pro odposlech obsahu komunikace, bakalářská práce, Brno, FIT VUT v Brně, rok 2012

Nástroj pro odposlech obsahu komunikace

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Libora Polčáka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Štefan Zima
16. Května 2012

Poděkování

Chtěl bych poďekovat svému vedoucímu Ing. Liboru Polčákovi, za rady a odbornou pomoc pri rešení této bakalářske práce.

Obsah

1 Úvod....2
1.1 Ciele práce....2
1.2 Obsah práce....3

2 Zákonný odposluch....4
2.1 Funkčné bloky referenčného modelu....5
2.2 Internal network interfaces (INI)....6
2.3 CC Trigger interface (CCTI)....6
2.4 CC Control interface (CCCI)....7
2.5 Handover interface (HI)....7
2.6 Prevádzka....8
2.7 Aplikovanie referenčného modelu....8

3 Zázemie pre zachytávania paketov....9
3.1 Architektúra koncového bodu....9
3.2 Operačný systém....10
3.3 Sieťový podsystém Linuxu....12
3.4 Paketové zachytávače....14
3.4.1 Knižnica libpcap....14
3.5 Problémy pri zachytávaní paketov....15
3.5.1 Kopírovanie paketov....15
3.5.2 Zátaž prerušeniami....15

4 Techniky urýchlenia prenosu....16
4.1 Eliminácia nadbytočných kópií....16
4.2 NAPI....17
4.3 Hybridné schémy....18
4.4 Zero-Copy....19
4.5 PF_RING....19

5 Vývoj aplikácie....22
5.1 Softvérové a hardvérové vybavenie....22
5.2 Špecifikácia požiadavkov....22
5.3 Analýza požiadavkov....23
5.4 Návrh....23
5.4.1 Odposluchávaný subjekt....24
5.4.2 Klient....24
5.4.3 Server....27
5.5 Implementácia....29
5.6 Profilovanie implementovaného riešenia....32

6 Testovanie....33

7 Záver....34

Zoznam príloh....38

1 Úvod

Počítačové siete sa stali neoddeliteľnou súčasťou nášho života. Široká škála služieb poskytovaných prostredníctvom Internetu je dnes v určitej podmnožine zahrňujúcej napríklad Webové služby, braná ako samozrejmosť pre narastajúci počet užívateľov. Práve to má za následok neustály vývoj nových sieťových prvkov a zlepšovanie vlastností prenosových liniek.

Pri čoraz väčšom dôraze na kvalitu poskytovaných služieb je nutné zaistiť krok s týmto vývojom aj na strane implementácie aplikácií. Dostatočné výkonným hardvérom nedosiahneme vždy požadované vlastnosti pri zle navrhnutej a implementovanej aplikácii, ktorá vykonáva neefektívne a časovo náročné operácie, v prípade Webového servera poskytovanie Webových stránok. Práve rýchlosť odozvy pri prenose sa stala pre bežných užívateľov jedným zo základných aspektov pre hodnotenie kvality.

Rast počítačovej kriminality v posledných rokoch a nove legislatívne ustanovenia o poskytovaní internetových a telekomunikačných služieb spôsobili zvýšenú pozornosť u správcov poskytovateľov týchto služieb. Prenosom citlivých informácii či zavedením služieb elektronického bankovníctva sa bezpečnosť počítačových sietí stala jednou z hlavných otázok, ktorá si vyžadovala nepretržitú pozornosť. Mnohé techniky pri návrhu zabezpečovacích mechanizmov nie vždy dostačujú pre ochranu sieťovej infraštruktúry.

Ako už bolo spomenuté, bezpečnosť je ďalším dôležitým zdrojom pozornosti pri procese vývoja služieb, kedy dochádza k možným následkom získaním a zneužitím prenášaných informácii. Osoby vedome páchajúc trestnú činnosť používajú sofistikovanejšie metódy prí získavaní týchto informácii a nie je vždy možné ich jednoduchým spôsobom vystopovať. Často používajú voľne dostupné služby umožňujúce skryť ich zámery, čím sa stali súčasťou diskusií na téma bezpečnosť počítačových sietí. Preto sa niektoré štáty rozhodli prijať ustanovenia, pomocou ktorých je možné tieto osoby usvedčiť.

Dokazovanie sa odvíja často od získaných dôkazov týkajúcich sa prenosu po sieti od útočníka alebo osoby vykonávajúcej činnosť v rozpore so zákonmi. Pri narastajúcich rýchlostiach prenosových liniek a počte užívateľov býva získavanie týchto dôkazov problémom. Tieto prenosy mnohonásobne prekračujú schopnosti získavať informácie prostredníctvom nástrojov využívaných pri odposluchu týchto osôb. Vývoj takýchto nástrojov je témou tejto práce.

1.1 Ciele práce

Táto práca sa týka oblasti počítačových sietí a rozoberá jednotlivé etapy vývoja nástroja, ktorý umožňuje zachytiť prebiehajúcu komunikáciu od odposluchávaného subjektu. Hlavným kritériom týchto nástrojov je schopnosť prijímať a zaznamenávať veľké dátové toky čo najväčšou možnou rýchlosťou s minimálnou stratou paketov. Rýchlosť, ktorú je možné dosiahnuť, sa odvíja od rýchlosti prenosovej linky čo zahrňuje použité médium, prenosový protokol a hardvérové vybavenie.

Pre návrh a implementáciu aplikácie na odposluch je nutné sa oboznámiť s princípmi fungovania systému pre zákonné odposluchy, procesom prijímania paketov v sieťovom podsystéme operačného systému Linuxového typu a nakoniec technikami umožňujúcimi zrýchlenie a zefektívnenie spracovania prichádzajúcich paketov. Vo výsledku využijeme jednu z techník, na základe ktorej implementujeme nástroj pre zachytávanie komunikácie. Nástroj využívajúci vybranú techniku otestujeme a porovnáme vlastnosti využitej techniky na základe nameraných výsledkov. V závere práce vyhodnotíme dosiahnuté výsledky a odvodíme kvalitu vytvoreného produktu. Kvalitou sa v tomto prípade myslí dosiahnutie nízkej stratovosti paketov, nízkej záťaže na výpočtový výkon hardvérového a softvérového vybavenia a bezstratové ukladanie zachytených dát.

1.2 Obsah práce

Práca je členená do niekoľkých hlavných kapitol zaoberajúcimi jednotlivými etapami vývoja aplikácie pre odposluch komunikácie. Nasledujúca kapitola 2, pojednáva o normách a štandardoch pre zákonne odposluchy. Uvádza referenčný model odposluchu a popisuje jednotlivé časti dôležité pre fázu analýzy požiadavok na výsledný nástroj. Kapitola 3 predstavuje teoretický úvod pre návrh a implementáciu nástroja pre odposluch komunikácie. Rozoberá všetky súčasti systému, na ktorom bude aplikácia spúšťaná a problémy, ktoré je nutné eliminovať technikami popisovanými v kapitole 4. Kapitola 5 obsahuje popis návrhu a implementácie aplikácie na základe poznatkov z kapitol 3 a 4. Fáza testovania v kapitole 6 hodnotí navrhnutý nástroj prostredníctvom série testov využitím hardvérových a softvérových generátorov prenosu. Zhodnotenie získaných poznatkov z návrhu a implementácie popisuje záverečná kapitola 7.

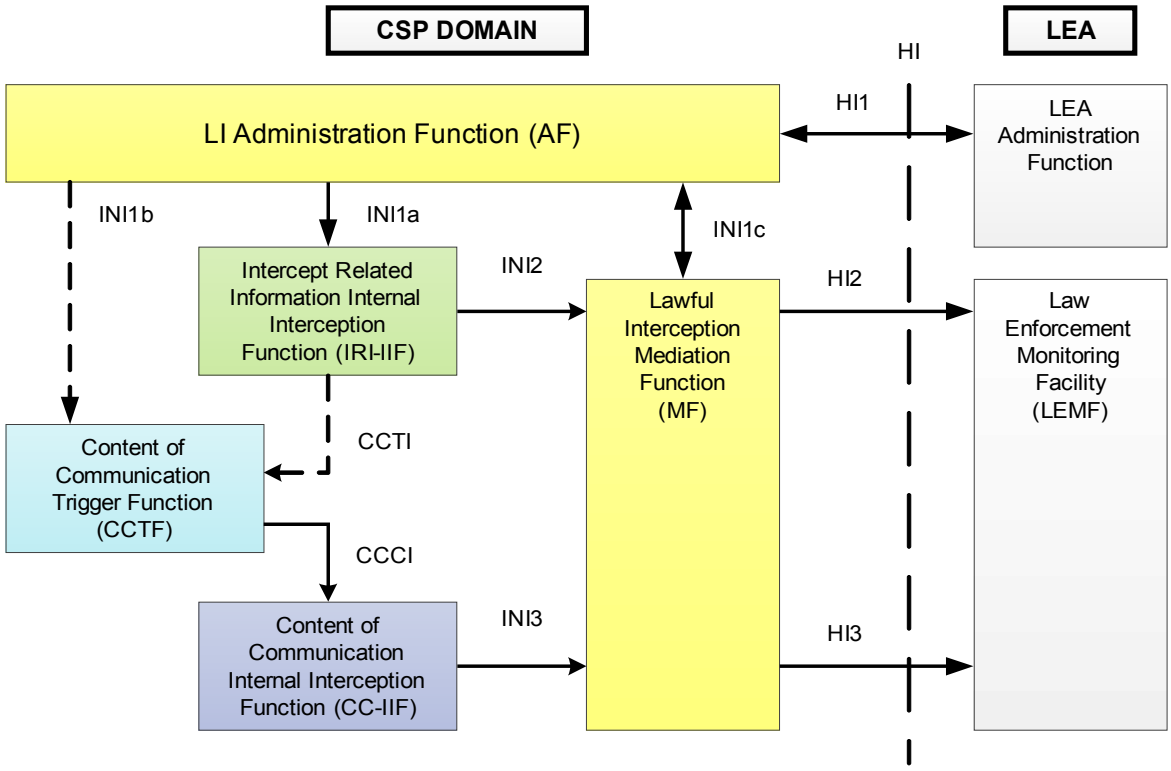
2 Zákonný odposluch

Termín *odposluch* existuje už od počiatku elektronickej komunikácie [1]. Ako právne prípustný, oficiálny prístup k súkromnej komunikácii, zákonný odposluch (LI) predstavuje bezpečnostný proces, v ktorom poskytovateľ služieb alebo sieťový operátor, zhromažďuje a poskytuje úradníkom činným v trestnom konaní zadržaný obsah komunikácie súkromných osôb alebo organizácii [4]. Táto definícia je súčasťou štandardu Európskeho Telekomunikačného Štandardizačného Inštitútu (ETSI), ktorý zaisťuje systematické postupy pri vykonávaní odposluchu.

Zákonný odposluch predstavuje získavanie dát z komunikačnej siete na základe požiadavku zákonnej autority alebo orgánov činných v trestnom konaní, pre účely analýzy komunikácie odposluchávaného subjektu a evidencie. Konkrétny obsah komunikácie je vyžadovaný len v niekoľkých prípadoch, prevažne pri podozrení z páchania trestnej činnosti, či narušenia kybernetickej bezpečnosti alebo ochrany sieťovej infraštruktúry.

Zákonný odposluch môžeme rozdeliť na aktívny a pasívny. Aktívny riešenie odposluchu predstavuje sieťové prvky schopné poskytovať rozhranie a funkcie, cez ktoré možno sprístupniť aktuálne zaznamenávane informácie. Naopak pasívny odposluch predstavuje strategicky umiestnené záznamové zariadenie zhromažďujúce dáta od užívateľa určeného pre odposluch.

Zákonné odposluchy sú iniciované zo strany organizácii činných v trestnom konaní (LEAs), v súlade s platnými zákonmi danej krajiny, od ktorých je možné žiadať platné povolenie počas prijímania a spracovávania odposluchu v reálnom čase. Tieto organizácie sú súčasťou nasledujúceho obrázku 2.1 popisujúceho referenčný model zákonných odposluchov pre siete využívajúce internetový protokol (IP).



Obrázok 2.1: Referenčný model pre zákonný odposluch [2]

Obrázok 2.1 popisuje základný referenčný model pre zákonný odposluch na základe ktorého budeme navrhovať aplikáciu. Tento model je obsiahnutý v odkazovanej norme a popisuje jednotlivé súčasti

zúčastňujúce sa na procese zákonného odposluchu so získavaním informácii o danom prenose a jeho konkrétnom obsahu.

Základný prvok, samotný obsah komunikácie (CC), je informácia vymieňaná medzi dvoma alebo viacerými užívateľmi v rámci telekomunikačnej siete. Tento obsah predstavuje odposluchávané dáta, ktoré sú špecifikované informáciou súvisiacou s odposluchom (IRI). IRI obsahuje kolekciu informácii a dát spájaných s komunikačnou službou, zahrňujúc identifikáciu cieľa špeciálne v súvislosti s hovorom, informácie alebo dáta ako napríklad neúspešný pokus o spojenie, informácie alebo dáta spojené so službou a informácie o mieste odkiaľ bol prenos zahájený.

Pred samotným zahájením tohto procesu je nutné vysvetliť niekoľko pojmov v samostatných podkapitolách zaoberajúcich sa funkčnými blokmi a rozhraniami modelu. Na základe týchto informácii bude možné popísať výmenu informácii medzi funkčnými blokmi a celkový proces zákonného odposluchu.

2.1 Funkčné bloky referenčného modelu

Každá sieť poskytovateľa služieb by mala obsahovať administratívnu funkciu (AF) pre správu žiadostí na odposluch. Tá zaisťuje, že sú poskytované informácie o CC, IRI alebo obidve, pre zber dát zo siete, ktoré sú následnej doručené do donucovacieho nástroja pre monitorovanie (LEMF), označovaného ako cieľ prenosu odposluchnutých informácii od odposluchávaného subjektu. Informácia dostupná v AF zahrňuje [2]:

- Identifikátor odposluchávaného subjektu.
- Dohodnutý identifikátor zákonného odposluchu (LIID).
- Začiatok a koniec alebo začiatok a dĺžka trvania odposluchu.
- Typ informácia zahrňujúci IRI alebo IRI aj CC.
- Adresu LFMF, na ktorú máju byť poslané IRI dáta.
- Adresu LFMF, na ktorú majú byť poslané CC dáta.
- Ďalšie informácie a voľby súvisiace s odposluchom.
- Referencia na autorizáciu odposluchu.

Vnútorná funkcia odposluchu pre informácie súvisiace s odposluchom (IRI-IIF) poskytuje IRI daného subjektu, ktorý je identifikovaný organizáciou LEA. Unikátny identifikátor subjektu je získaný prostredníctvom AF.

Vnútorná funkcia odposluchu pre obsah komunikácie (CC-IIF) má za úlohu presunúť CC do sprostredkovateľskej funkcie (MF), ktorá uskutočňuje koreláciu a formátovanie IRI a CC pre doručenie do LEMF.

Zámerom spúšťacej funkcie odposluchu komunikácie (CCTF) je identifikácia zariadenia CC-IIF a jeho ovládanie prostredníctvom kontrolného rozhrania (CCCI).

Na výmenu informácii medzi jednotlivými funkčnými blokmi slúžia vnútorné rozhrania siete (INI), odovzdávajúce rozhrania (HI), spomínané rozhranie CCCI a rozhranie pre spúšťanie odposluchu (CCTI). Tie sú popísané v nasledujúcich podkapitolách.

2.2 Internal network interfaces (INI)

Vnútorné rozhrania INI1 až INI3 prenášajú počiatočné informácie ohľadne zákonného odposluchu. Rozhranie INI1 je rozdelené na tri časti označené INI1a, INI1b a INI1c.

- INI1a využíva AF a má za úlohu poskytnúť IRI pre IRI-IIF.
- INI1b využíva AF pre poskytnutie informácií CCTF týkajúce sa konkrétneho odposluchu.
- INI1c využíva AF pre poskytnutie informácii MF, na základe ktorých vytvára koreláciu CC a IRI.

Ak by sme uvažovali IIF ako IRI-IIF a CC-IIF v jednom bloku, informácie poskytované prostredníctvom interných rozhraní INI1a až INI1c obsahujú [2]:

- Identifikátor odposluchu LIID.
- Informácie o odposluchu, z pohľadu rozhrania INI1a ide o unikátny identifikátor cieľa odposluchu a z pohľadu rozhrania INI1b špecifikáciu pre CC-IIF, ktorá predstavuje filtre pre samotný obsah komunikácie CC. Tá môže obsahovať cieľovú alebo zdrojovú IP adresu, port či iné parametre jednoznačne identifikujúce cieľový subjekt.
- Cieľové adresy pre preposielanie IRI a CC do MF.
- Parametre pre zapuzdrenie a transport IRI a CC dát.
- Informácie pre splnenie bezpečnostných požiadaviek MF.

Vnútorné rozhranie INI2 má za úlohu preposielať informácie súvisiace s IRI vyžadované od LEMF do MF. Tie sa týkajú konkrétnych záznamov štruktúry IRI. Keďže MF vykonáva koreláciu dát od CC-IIF a IRI-IIF, IRI-IIF sprostredkováva informáciu o identifikátore odposluchu LIID a korelačne informácie z IRI.

INI3 je vnútorne rozhranie, prostredníctvom ktorého zasiela CC-IIF dátové obsahy komunikácie na ďalšie spracovanie do MF.

Bez ohľadu na použitú metódu zapuzdrenia pre prenos do MF by prenášaná informácia mala zostať nezmenená so zachovaním hlavičiek protokolov aj samotným obsahom. Pre koreláciu s IRI v MF by mala tak isto poskytovať identifikátor odposluchu. Úlohou tohto rozhrania je tiež poskytovať jednoduchý implementačný mechanizmus, ktorý nevýrazne ovplyvňuje sieťové elementy a zaisťuje zachovanie kvality služieb (QoS[1]) a bezpečnosť.

2.3 CC Trigger interface (CCTI)

Prostredníctvom rozhrania CCTI posiela IRI-IIF informácie pre spustenie odposluchu. IRI-IIF cez toto rozhranie upovedomí CCTF, kde sa nachádza CC-IIF spojené s odposluchávaným subjektom a špecifikuje CC filtre pre danú komunikáciu. Tieto a ďalšie informácie vyžadované od CCTF by mali byť dostačujúce pre zahájenie komunikácie s CC-IIF.

1 *Quality of Service*, kvalita služieb, predstavuje riadenie a kontrolu prenosu pre zvýšenie kvality vybraných dátových tokov.

Ďalšie informácie by mali zahrňovať [2]:

- LIID identifikátor odposluchu.
- Špecifikáciu CC filtrov, obsahujúcu IP adresu, číslo portu a ďalšie identifikátory relácie odposluchávaného subjektu. Tieto informácie by nemali byť známe pred tým než subjekt zahájí komunikáciu.

Ak nie je k dispozícii informácia u umiestnení CC-IIF, zahájí CCTF dynamické prehľadávanie adresového priestoru siete alebo začne prehľadávať uloženú tabuľku adries.

2.4 CC Control interface (CCCI)

Po zistení umiestnenia CC-IIF, začne CCTF komunikovať prostredníctvom CCCI. Týmto rozhraním zahajuje odposluch na CC-IIF tým, že špecifikuje prenášanú informáciu podobne ako rozhranie INI1b obsahujúcu identifikátor LIID, CC filtre preposielané od IRI-IIF a informácie, kam má CC-IIF posielať CC. Tieto informácie zahrňujú cieľovú adresu MF pre príjem CC a IRI od CC-IIF a IRI-IIF.

Pri zlyhaní CCTF by mala CC-IIF prestať vykonávať odposluchy a byť schopná znovu ich ustanoviť pri znovu-spustení CCTF. To je možné zaviesť mechanizmom časovača, ktorý sa pri každom prijatí informácie o filtri CC aktivuje a pri dosiahnutí nastaveného časového limitu zruší v CC-IIF. Pri zlyhaní CC-IIF by sa mal prostredníctvom rozhrania CCCI znovunastaviť odposluch blokom CCTF.

2.5 Handover interface (HI)

Odovzdávacie rozhrania špecifikuje troj-portová štruktúra logicky oddeľujúca rozhranie pre administratívne informácie HI1, rozhranie pre informácie súvisiace s odposluchom HI2 a rozhranie pre obsah komunikácie HI3 [3].

Rozhranie HI1 prenáša rôzne druhy administratívnych informácií od a do LEA. Tvorí komunikačný kanál medzi AF a LEA a môže prekračovať hranice štátov na základe medzinárodných dohôd. Poskytuje prenos informácii medzi poskytovateľmi a operátormi telekomunikačných, internetových služieb a LEA, ohľadom stanovených odposluchov.

Rozhranie HI2 ma za úlohu prenášať informáciu od poskytovateľov a operátorov sietí IRI informácie o odposluchu do LEMF. Prenos týchto informácii by mal byť realizovaný spôsobom vhodným pre sieťovú infraštruktúru odvíjajúc sa od verzie a typu dát.

Posledným z trojice odovzdávacích rozhraní je rozhranie HI3. Jeho úlohou je prenos CC od MF do LEMF. Dáta prenášane týmto rozhraním by mali byť v zrozumiteľnom formáte bežného jazyka popisujúce ustanovenie spojenia, obojsmernú komunikáciu a samotný CC odposluchávaného subjektu. Ak CC obsahuje šifrovanú komunikáciu, obsah prenášaných dát musí byť dešifrovaný, prípadne musia byť dodané kľúče a dešifrovacie algoritmy pred dodaním do LEA. To sa deje len v prípade že je komunikácia šifrovaná v rámci sieťovej infraštruktúry a ktorej šifrovanie a dešifrovanie má na starosti operátor alebo správca siete. Povinnosť poskytnúť spomínané dešifrovacie algoritmy spolu s kľúčmi má na starosti poskytovateľ služieb alebo operátor siete. V prípade že komunikáciu šifruje odposluchávaný užívateľ, táto povinnosť operátora a poskytovateľa odpadá.

Rozhrania HI2 a HI3 sú v logicky rozdielne no v niekoľkých inštaláciách môžu predstavovať jeden prenosový kanál. V tomto prípade sú v správach odkazované dátové polia tokov CC-IIF a IRI-IIF.

2.6 Prevádzka

V bežne prevádzke [2], LEA vygeneruje požiadavok na odposluch špecifického subjektu, či už súkromnej osoby alebo organizácie a pošle ho prostredníctvom odovzdávacieho rozhrania HI1 do AF. Následkom toho autorizovaná osoba v rámci AF ustanoví odposluch na základe IRI alebo IRI a CC súčasne. INI rozhraniami postupne prepošle subjektom CCTF, IRI-IIF a MF informácie o zahajovanom odposluchu.

Zariadenie CCTF zaháji odposluch cez rozhranie CCCI na zariadení CC-IIF okamžite alebo až po prijatí spúšťacej správy rozhraním CCTI.

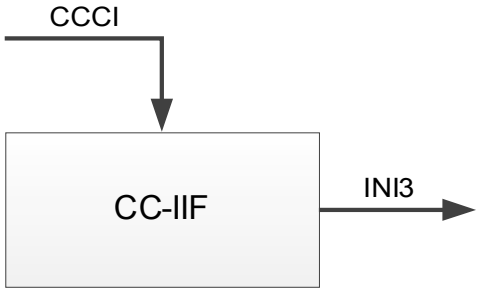
Po zahájení odposluchu začnú zaradenia CC-IIF a IRI-IIF posielať prostredníctvom rozhraní INI2 a INI3 správy CC a IRI do MF. Ak neboli pridané LIID informácie do správ počas spracovania v IRI-IIF a CC-IIF pridá ich MF a uskutoční koreláciu CC a IRI. Po pridaní identifikátora a korelácii prevedie správy do formátu pre prenos odovzdávacími rozhraniami pre LEMF.

Pri plánovaní odposluchu je nutné zvážiť možnosť šifrovania zo strany poskytovateľa a operátora telekomunikačných a internetových služieb alebo užívateľa. Vysporiadanie sa s tým problémom bolo popisované v podkapitole 2.5 na odovzdávacom rozhraní HI3, ktoré predpisuje formát správ v bežnom jazyku v zrozumiteľnej forme tak, že LEA ukladá povinnosť operátorovi alebo poskytovateľovi služieb zabezpečiť šifrovacie a dešifrovacie algoritmy spolu s kľúčmi.

Zátaž, ktorú produkuje spracovávanie zachytávaných dát by nemala ovplyvňovať ďalšie prenášane služby.

2.7 Aplikovanie referenčného modelu

Pre naše potreby je nutné sústrediť sa na funkčný blok CC-IIF, predstavujúci zariadenia prepojené s odposluchávaným subjektom. Vzhľadom na to, že komunikácia od odposluchávaného subjektu môže prebiehať pred ustanovením odposluchu, je nutné aplikovať CC filtrovacie pravidla počas prebiehajúcej komunikácie. Tie sú generované v časti IRI-IIF a cez CCTI rozhranie prenášane do funkčného bloku CCTF, ktorý ma na starosti ovládanie celého nástroja pre odposluch. Obrázok 2.2 znázorňuje časť referenčného modelu pre návrh nástroja pre odposluch.



Obrázok 2.2: Interná funkcia pre odposluch obsahu komunikácie [2]

3 Zázemie pre zachytávania paketov

V tejto kapitole sa pozrieme na základné komponenty využité pri realizácii programu na odposluch komunikácie. Vývoj podlieha niekoľkým fázam, pri ktorých je nutné zvážiť všetky vlastnosti a závislosti týchto komponentov pre odôvodnenie získaných výsledkov. V podkapitole 3.1 sa pozrieme na architektúru koncového bodu v sieti, na ktorom bude realizovaná implementácie našej aplikácie. Podkapitola 3.2 rozoberá použitý operačný systém a jeho vlastnosti nutné pre pochopenie procesu prijímania paketu popisovaného v podkapitole 3.3 na sieťovom podsystéme operačného systému Linuxového typu. Predposledná podkapitola 3.4 rozoberá existujúce aplikačné rozhranie (API) pre tvorbu nástrojov, ktoré zachytávajú pakety a načrtne fungovanie týchto nástrojov. V poslednej podkapitole 3.5 rozoberieme v krátkosti problémy, ktoré je nutné eliminovať pri tvorbe takýchto nástrojov.

3.1 Architektúra koncového bodu

Program bude implementovaný na osobnom počítači, ktorý spadá do triedy koncových zariadení z pohľadu počítačových sietí. Koncové body v sieťach, sú predovšetkým zamerané na všeobecné výpočty narozdiel od aktívnych sieťových prvkov, akými sú napríklad smerovače alebo prepínače.

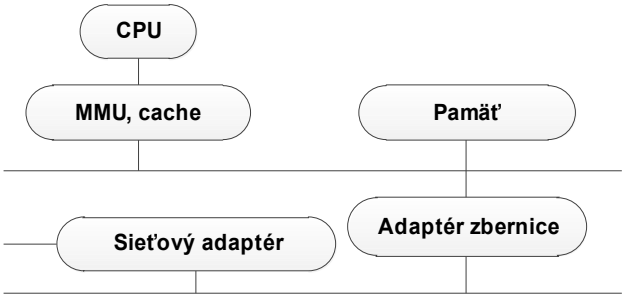
Koncový bod alebo inač povedané pracovná stanica, využíva procesor (CPU) a prácu s pamäťou pri vykonávaní inštrukcií. CPU môžeme nazvať stavovým automatom, ktorý vykonáva sekvenciu inštrukcií spolu s dátami ako vstup a zapisuje výstup do vstupno-výstupných (I/O) zariadení akými sú napríklad aj sieťové adaptéri [5]. Pri vykonávaní aplikácie, ktorá spotrebováva procesorový čas a pamäť, je časť stavu uložená v pamäti typicky označovanej RAM. Pri práci s touto pamäťou dochádza k spomaleniu vykonávania inštrukcií preto CPU obsahuje pamäte typu *cache* pre urýchlenie vykonávania inštrukcií. Prístup a správu operačnej pamäte má na starosti jednotka správy pamätí (MMU), ktorá je dnes už súčasťou CPU.

Väčšina I/O zariadení je mapovaných do pamäte. V tomto ponímaní pristupuje CPU počítača k jednotlivým zariadeniam podobne ako pri štandardnej práci s pamäťou. Pri vykonávaní I/O operácii, u ktorých participuje CPU, môže dôjsť k spomaleniu, keďže CPU čaká kým zariadenie napríklad zapisujúce dáta na zbernicu ukončí svoju činnosť.

Pri I/O operáciach, u ktorých participuje CPU, dochádza k spomaleniu vykonávania inštrukcií, keďže CPU musí počkať kým zariadenia dokončí zápis dát na zbernicu. Moderné zariadenia umožňujú priamy prístup (DMA) bez zásahu CPU čím urýchľujú zápis či čítanie z pamäti. Problémom je však že musia túto zbernicu s ostatnými zariadeniami, ktoré spotrebovávajú procesorový čas. Zatiaľ čo pamäťové zbernice sa vyvíjajú s každým novým procesorom, I/O zbernice využívajú spätnú kompatibilitu čím ich rýchlosť zostáva prevažne rovnaká a vývoj pomalší [5].

Moderné viac-vláknové architektúry CPU umožňujú oddeliť vykonávanie jednotlivých inštrukcií napríklad pri zápise do pamäte pri prijímaní paketov a ich spracovávaním. Tým dosiahnú zrýchlenie výpočtu napríklad pri spracovávaní avšak pri vzájomnej komunikácii s pamäťou pridávajú niektoré ďalšie problémy.

Znalosť práce s pamäťou je dôležitým predpokladom pri návrhu aplikácie sprácovávajúcej veľké paketové toky, čo môže výrazne ovplyvniť celkový výkon nielen aplikácie aj celkového systému. Na obrázku 3.1 je zobrazená jednoduchý model pracovnej stanice prevzatý z [5].



Obrázok 3.1: Model pracovnej stanice [5]

3.2 Operačný systém

Väčšina koncových zariadení či už klientov alebo serverov používa operačný systém. Ten je umiestnený nad hardvérovou vrstvou čím umožňuje jednoduchšiu prácu programátorov aplikácii. Keďže samotné operácie pri práci s hardvérom sú náročné ako napríklad práca s pamäťou, ovládanie prerušení a I/O operácie, poskytuje operačný systém istú mieru abstrakcie. Tá predstavuje zjednodušenie pohľadu na tieto operácie ako na systém bez prerušení, prácu so súvislou oblasťou pamäti alebo jednoduchosť I/O operácii.

Každá z týchto abstrakcií predstavuje riziko pri návrhu a implementácie aplikácie, keďže sú tieto prostriedky v réžii jadra, ktoré ukrýva implementačné detaily spomínaných operácii abstrakciou. Týmto môže programátor predpokladať umiestnenie dátovej štruktúry na predpokladané miesto v pamäti pričom operačný systém môže využiť úplné iné umiestnenie.

Vykonávanie procesov je plánované častou systému zvanou plánovač. Za predpokladu, že proces aplikácie je vykonávaný nepretržite však môže dôjsť k prerušeniu od hardvérového zariadenia, čím sa vykonávanie presunie. Operačný systém uloží stav procesu do pamäte. Nasleduje vykonávanie obsluhy prerušenia, ktorá má vyššiu prioritu. Po jej vykonaní plánovač obnoví stav procesu a pokračuje jeho vykonávanie. Pri tomto prebieha k niekoľkým zložitým operáciám ako je prepínanie kontextu procesu, jeho plánovanom spustení a ochrany pred ovplyvnením vykonávania ďalších procesov.

Proces ako vykonávaná entita v operačnom systéme môže predstavovať pri varianty. Prvou z nich je už spomínaná obsluha prerušení, ktorá slúži na rýchle výpočty využívajúc malý stav systému v podobe registrov. Vykonáva prevažne atomické operácie.

Ďalšiu tvoria užívateľské procesy. Narozdiel od prerušení využívajú celkový stav systému zahrňujúc hlavnú pamäť a registre. Prepínanie kontextu vzhľadom k tomu môže predstavovať časovo náročnú operáciu vykonávanú plánovačom.

V rámci kontextu procesu, vlákna poskytujú jednoduchšiu a rýchlejšiu variantu oproti klasickým procesom. Pri výtvarní nového procesu v rámci aplikácie je nutné premapovať celú oblasť pamäte využívajúcu procesom, ktorý vytvára potomka. Narozdiel toho vlákna, nevyžadujú náročne mapovanie pamäte keďže tá je s procesom v ktorom boli vytvárané zdieľaná. Hlavný proces v tomto prípade predstavuje hlavné vlákno *(main thread)*.

Napriek mnohým výhodám prinášajú vlákna aj problémy v podobe ich vzájomnej synchronizácie s procesom.

Postup prijímania paketu z pohľadu prerušení a plánovania procesov vyzerá nasledovne:

1. Paket, ktorý dorazí na rozhranie sieťovej karty generuje hardvérové prerušenie.

2. Procesor uloží stav práve prebiehajúceho procesu.

3. Procesor skočí na vykonávanie obsluhy prerušení a vynechá plánovač.

4. Obsluha prerušenia prekopíruje paket do rady IP paketov v jadre a zavolá vlákno operačného systému. To predstavuje softvérové prerušenie, ktoré má vyššiu prioritu než plánovač.

5. Obsluha prerušenie odovzdá riadenie plánovači a skončí. Spomínaná vyššia priorita softvérového prerušenia spôsobí prenechanie kontroly plánovačom tejto obsluhe.

6. Obsluha softvérového prerušenia spracuje paket na úrovni transportného protokolu (TCP) a internetového protokolu (IP).

7. Paket následne presunie do aplikačnej rady nazývanej *rada schránky.*

Ďalšia kapitola bude rozoberať tieto kroky podrobnejšie a operačnom systéme Linux. Linux, je slobodný (*free*) operačný systém, čo znamená že jeho zdrojový kód je dostupný komukoľvek. Táto vlastnosť mu umožnila rozšírenie do oblasti komerčného využitia u sieťových prvkov. Spoločnosti produkujúce sieťový hardvér v nemalej miere využívajú tento systém ako nízko-úrovňový operačný systém pre svoje produkty. Rozšírenie zahrňuje aj osobné počítače a serverové riešenia z dôvodu jednoduchej správy a prostriedkov nielen pre tvorbu efektívnych sieťových aplikácii.

V tejto súvislosti sa zameriame práve na vývoj pod týmto operačným systémov bežiacom na bežne dostupnom hardvéri.

Z hľadiska abstrakcie predstavujúcej nekonečnú pamäť, programátor predpokladá použitie neprerušeného úseku pamäte pre svoju aplikáciu. Táto pamäť je však virtuálna a mapovaná do fyzickej časti hlavnej pamäte a prípadne do pamäte pevného disku. Pre uskutočnenie tohto mapovania sa predpokladá že aplikácia pri spustení alokuje pre svoj chod súvislú oblasť vo virtuálnej pamäti. Požadovaný úsek pamäti je limitovaný veľkosťou fyzickej pamäte z čoho vychádza jeden z problémov. Ten je možné odstrániť mechanizmom *mapovania stránok na základe tabuľky stránok* a *stránkovania na vyžiadanie.* Obe metódy mapujú pre aplikáciu virtuálne adresový priestor do fyzického. Ten ako už bolo spomínané, môže byť vytvorený v hlavnej pamäti alebo na disku.

Mapovanie na základe tabuľky stránok rozdeľuje adresové značenie virtuálnej pamäte na časť reprezentujúcu umiestnenie vo fyzickej pamäti a časť na umiestnenie v rámci danej stránky. Pri vyžiadaní stránky, ktorá sa nachádza na disku vyvolá hardvér výnimku a operačný systém pre-mapuje stránku z disku do fyzickej pamäte pre rýchlejší prístup [5]. Toto popisuje druhý prístup pri mapovaní stránok na vyžiadanie. Mapovanie stránok môže byť časovo náročná operácia, preto aktuálne mapované stránky moderné procesory ukladajú do cache pamäte nazývanej *Translation Lookaside Buffer* (TLB) [5]. Samotné mapovanie má na starosti spomínaná jednotka MMU.

Mapovanie náročných I/O operácii na fyzické operácie ako je napríklad `read()` vykonávajú ovládače zariadení. Ak by jediným problémom bola abstrakcia, bolo by možné umiestniť zdrojové kódy ovládačov do knižníc prístupných aplikáciam [5]. Problémom je využitie zariadení viacerými procesmi, čo by mohlo predstavovať riziko ak by sa poškodený proces pokúšal o prístup k zariadeniu čím by mohol narušiť jeho funkciu alebo ho poškodiť. Preto sú uskutočňované systémové volania funkcií komunikujúcimi so zariadeniami v chránenej sekcii jadra operačného systému. Vykonávanie týchto operácii môže byť časovo náročné (niekoľko mikrosekúnd), čo predstavuje výrazne spomalenie aj u rýchlejších procesorov [5].

3.3 Sieťový podsystém Linuxu

Obrázok 3.2 zachycuje spracovávanie proces prijímania paketov sieťovým podsystémom Linuxu so spracovaním TCP. Poukazuje na cestu od prenosovej linky až po aplikačnú oblasť pamäte.

Jadro operačného systému Linux využíva štruktúru *sk_buff* pre každý prijatý paket do veľkosti maximálnej prenosovej jednotky (MTU) siete [7]. Tieto štruktúry sú uložené v kruhovej vyrovnávacej pamäti nazývanej *rx_ring* kontrolovanej ovládačom sieťovej karty a musia byť v čakajúcom stave pripravenom na príjem paketu, *ready.* Tento stav špecifikuje inicializovanú a alokovanú štruktúru sk_buff namapovanú do I/O pamäťového priestoru pre DMA operácie.

Keď na rozhranie sieťovej karty (NIC) dorazí paket, ovládač aktivuje DMA nástroj, ktorý prekopíruje paket do tejto štruktúry. V súvislosti s prekopírovaním paketu, nastaví DMA príznak tejto štruktúry informujúci CPU, že daný deskriptor sk_buff je použitý. Veľkosť kruhovej vyrovnávacej pamäte rx_ring závisí od konkrétnej sieťovej karty a jej ovládača. Po jej zaplnení, sú ďalšie pakety zahadzované, preto pri spracovaní každého paketu v tejto vyrovnávacej pamäti je nutné použitú štruktúru sk_buff uvoľniť a znovu alokovať a inicializovať.

Po tom čo je paket úspešne presunutý do hlavnej pamäte a je prístupný jadru, vyvolá ovládač prerušenie pre upovedomenie CPU, ktorého reakciou je spustenie obsluhy prerušenia ovládača a naplánovanie *softvérového prerušenia.* Obsluha tohto prerušenia vloží do rady pre dotazovanie CPU identifikátor rozhrania a zablokuje prerušenia od sieťovej karty pri príchode ďalších paketov.

Počas softvérového prerušenia CPU spracováva pakety každého rozhrania uloženého v rade pre dotazovanie. Dotazuje sa na ne metódou `poll()`.
Zatiaľ, čo je paket spracovávaný na vyšších vrstvách protokolového zásobníku, zostávajú jeho dáta v pamäti jadra pre zamedzeniu nadbytočných kópií [6].

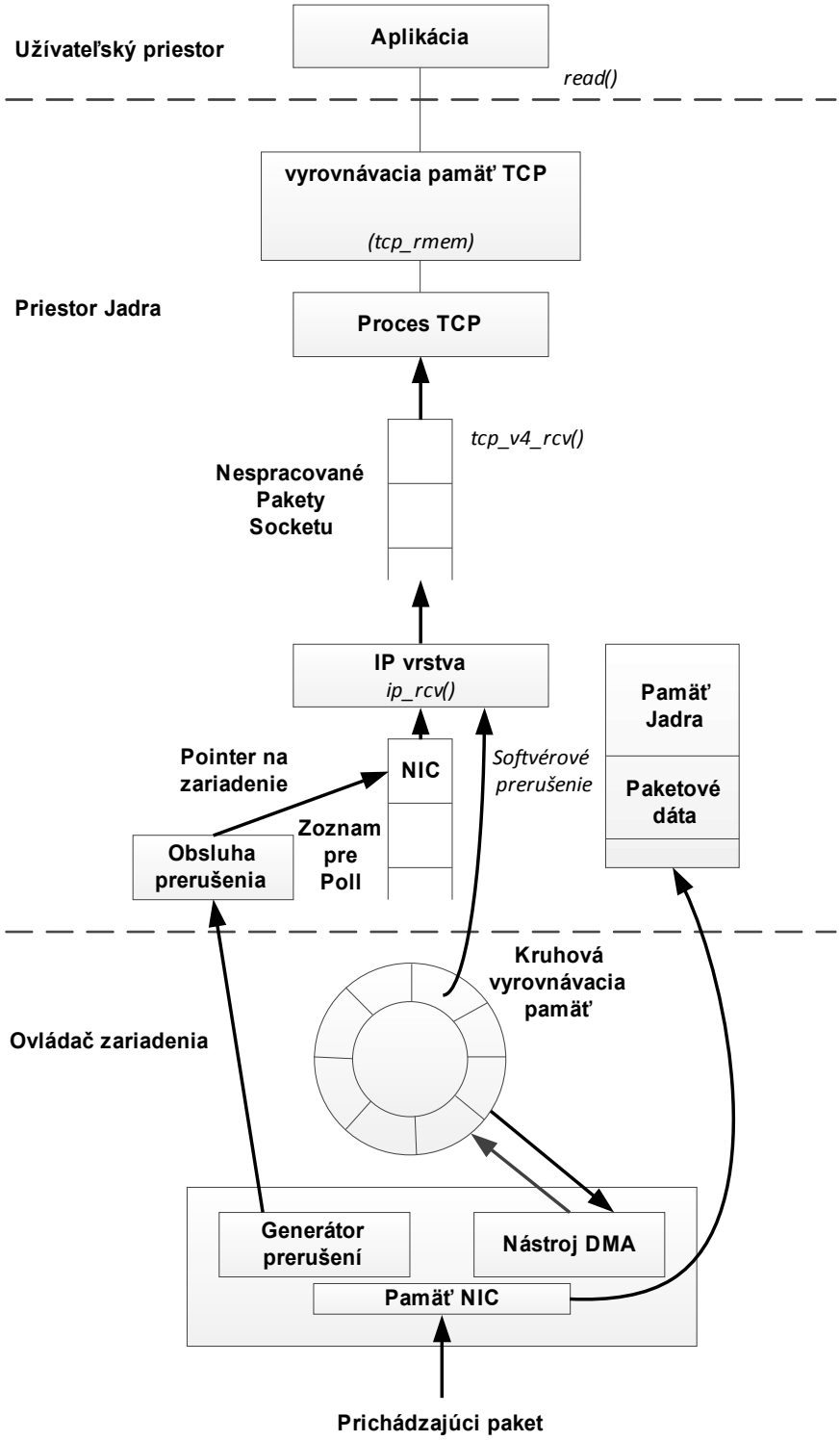
Ako už bolo spomenuté, fázu prijímania paketov možno rozdeliť do troch etáp [7]:

1. Prijatý paket je presunutý z NIC do kruhovej vyrovnávacej pamäte rx_ring. Tento proces riadi a kontroluje ovládač sieťovej karty.

2. Paket je následne presunutý z tejto pamäte, do vyrovnávacej pamäte schránky. Toto kopírovanie je reakciou na softvérové prerušenie naplánované obsluhou prerušenia ovládača. Proces je riadený protokolovým zásobníkom jadra.

3. Nakoniec sú dáta paketu kopírované z vyrovnávacej pamäte schránky do vyrovnávacej pamäte aplikácie. Je to proces prijímania dát.

Spracovávanie v druhej fáze závisí na použitých protokoloch. Spracovanie na IP vrstve ja zahájené volaním funkcie `ipv4_rcv()`. Po preskúmaní hlavičky protokolu IP je determinovaný protokol vyššej vrstvy. Pri protokole TCP, sú vytvárané rady pre zachovanie toku v prípade chýb a nepotvrdených paketoch a až po následných opravách sú dáta prijímané pamäťou schránky aplikácie. Pakety sú prijímané volaním funkcie `tcp_v4_recv` v prípade protokolu TCP a funkciou `upd_recv` v prípade protokolu UDP. Dáta transportnej protokolovej vrstvy sú kopírované prostredníctvom štruktúry *iovec.*



Obrázok 3.2: Príjem paketu sieťovým podsystémom Linuxu[6]

3.4 Paketové zachytávače

Predtým než sa pustíme do návrhu a implementácie našej aplikácie je nutné oboznámiť sa s existujúcimi aplikáciami pre zachytávanie paketov označovanými ako *Packet Snifffers* (PS). Jedným z príkladov takýchto aplikácii je aj *tcpdump* [9].

Táto aplikácia slúži na záznam a monitorovanie prenosu na sieti prostredníctvom sieťovej karty a knižnice *libpcap* [8] navrhnutej v rámci vývoja tohto programu.

Paketový zachytávač (PS) je aplikácia, umiestnená na sieťovom zariadení, zachytávajúca všetky rámce linkovej vrstvy posúvané cez sieťové rozhranie [10].

Na to aby sieťová karta prijímala pakety, ktoré nie sú určené pre ňu, je nutné ju prepnúť do *promiskuitného* módu. Ak by v tomto móde zapnutá nebola, kontrolovala by *ethernetovú* [13] hlavičku každého rámcu prijatého zo siete. V prípade žeby cieľová MAC [11] adresa nezodpovedala MAC adrese sieťovej karty, na ktorú rámec dorazil, zahodí ho. Naopak v promiskuitnom móde prijíma rámce, ktoré nemusia byť určené práve jej.

Pre analýzu paketu musím rozdeliť paket do protokolových vrstiev OSI modelu [12]. Pre rozpoznanie protokolov vyšších vrstiev a výpisu informácii o zachytenom pakete je nutné poznať štruktúry protokolov a ich hlavičky. Tie sú umiestnené v typicky v hlavičkových súboroch, v operačnom systéme Linux v adresári */usr/include/netinet/* alebo */usr/include/linux/*.

3.4.1 Knižnica libpcap

Pre realizáciu základných nástrojov na zachytávanie paketov slúži štandardná knižnica libpcap. Tá poskytuje aplikačné programovacie rozhranie (API) pre implementáciu týchto nástrojov. Libpcap je knižnica napísaná v jazyku C, poskytujúca niekoľko funkcií pre ovládanie sieťovej karty. Umožňuje získať prijaté pakety priamo zo sieťovej karty [10]. Zavádza nový typ schránky *PF_PACKET*, ktorá obchádza spracovávanie paketu jadrom.

Keďže samotné zachytávanie paketov sa uskutočňuje na úrovni jadra, spracovávanie prebieha na strane aplikácie v užívateľskom prostredí a pri veľkom prenose dochádza k opozdeniu spracovávania paketov čo má za následok stratu niektorých paketov.

Knižnica libpcap umožňuje na schránku PF_PACKET umiestniť filtrovacie pravidlá označované LPF (*Linux Packet Filter*) [10], ktoré zahadzujú nežiadúce pakety a tým ponúkajú možnosť urýchliť spracovanie vybraných paketov. Aplikácia filtrovacích pravidiel musí prebiehať pre zahájením zachytávania paketov.

Spomedzi všetkých funkcií knižnice sú tieto najpodstatnejšie:

- `pcap_open_live()` ako už jej názov napovedá, vracia štruktúru typu `pcap`, `pcap_t` a otvára novú reláciu zachytávania paketov pre stanovené parametere. Vrátená štruktúra je používaná v celom programe pre ďalšie spracovanie.

- `pcap_next()` a `pcap_loop()` sú funkcie realizujúce samotné zachytávanie paketov. Rozdiel medzi týmito funkciami spočíva v spôsobe získavania paketov z rozhrania. Funkcia `pcap_next()` sa obyčajne volá v cykle pre každý prijatý paket. Naopak funkcia `pcap_loop()` tento cyklus implementuje vnútorne čím je možné ju ukončiť len volaním funkcie `pcap_breakloop()`. Obe tieto funkcie disponujú parametrom špecifikujúcim procedúru, ktorá je volaná pre každý prijatý paket na spracovanie. Tá má predpísaný formát obsahujúci parametre prijatého paketu a užívateľských dát pre presun parametrov do procedúry.

- `pcap_setfilter()` funkcia je použitá na spomínané aplikovanie filtrov pre deskriptor PF_PACKET. Využíva pritom filtrovacie pravidlá BPF (*Berkeley Packet Filter*) [9], ktorých špecifikáciu možno nájsť v manuálovej stránke aplikácie tcpdump. Tieto pravidlá sú zapísané formou reťazca znakov, ktorý je nutné preložiť funkciou `pcap_compite()` a aplikovať na deskriptor.

Pre ďalšie informácie a popis funkcií knižnice libpcap používame manuálovú stránku `pcap`.

3.5 Problémy pri zachytávaní paketov

V predchádzajúce kapitole 3.1 popisujúcej knižnicu libpcap bolo spomenuté spracovávanie veľkého počtu paketov, ktoré spôsobuje problémy prevažne na strane aplikácie v užívateľskom priestore. Operačné systémy poskytujúc abstrakciu popisovanú v kapitole 3.2 vytvárajú takzvanú vrstvovú architektúru (*layered software*) [5] pre zabránenie aplikáciam narušiť chod celého systému. To sa odvíja od použitia protekčných domén (*procetcion domain*), naprieč ktorými kopírovanie predstavuje výrazne spomalenie aj s využitím rýchlejšie hardvéru.

Operačné systémy poskytujú všeobecne navrhnuté funkcie pre použitie I/O operácií tak aby pokryli čo najširšiu oblasť vytváraných aplikácii. To predstavuje pre programátora abstrakciu s predpokladom rýchleho spracovania, ktoré sa však odvíja najmä od rýchlosti I/O zbernice a prístupovej doby k zariadeniam akými sú napríklad pevné disky.

3.5.1 Kopírovanie paketov

Pri spracovávaní paketu na úrovni jadra cez rôzne protokolové sekcie vzniká veľké množstvo kopírovaní, ktoré je možné eliminovať technikami popisovanými v kapitole 4. Aplikácie vyžadujúce spracovanie veľkého počtu paketov spomaľuje kopírovanie paketov z pamäte sieťovej karty do hlavnej pamäte jadra a následne do pamäte samotnej aplikácie. Tieto reštrikcie spôsobujú záťaž v podobe už spomínaných systémových volaní.

Pevné disky a zbernice ako napríklad PCI [14] patriace medzi najpomalšie hardvérové súčasti systému spôsobujú pri dnešných rýchlostiach prenosových liniek dosahujúcich gigabitové rýchlosti výrazne spomalenie aplikácii nielen pri kopírovaní ale aj pri spotrebe procesorových cyklov.

3.5.2 Zátaž prerušeniami

Veľký tok dát prichádzajúci na sieťové rozhranie spôsobuje neustále vyvolávanie prerušenia od ovládača sieťovej karty. Aj napriek tomu, že obsluha prerušenia nepredstavuje zložité operácie vykonávané samotnou aplikáciu, pri veľkom počte môže spotrebovať celkový výkon a procesorový čas natoľko, že dôjde zamedzeniu vykonávanie aplikácie. Tento prípad sa označuje termínom vyhladovanie (*starvation*). Po zaplnení vstupnej rady paketov dochádza k ich vyhadzovaniu, keďže aplikácia nestihne vykonať spracovanie. Pre celý systém prijímajúci pakety dochádza k prechodu do stavu uviaznutia (*receiver livelock*) [5].

4 Techniky urýchlenia prenosu

Táto kapitola popisuje rôzne techniky urýchlenia spracovania prichádzajúcich paketov. Zameriavajú sa na kopírovanie medzi jadrom operačného systému a užívateľským prostredím,v ktorom beží aplikácia. Podkapitola 4.1 je zameraná rôzne druhy kopírovania vo všeobecnosti, demonštrovaných na Webovom serveri [5]. Nasledujúce podkapitoly rozoberajú konkrétne techniky obchádzania spracovania paketov v sekcii jadra.

Kapitola 4.2 popisuje mechanizmus NAPI [17], ktorý eliminoval problém zahltením prerušení od sieťovej karty pri prijímaní veľkého počtu paketov z dátového toku. NAPI ako aj ďalšie hybridné architektúry kombinujúce viaceré techniky pri spracovávaní paketov sú predmetom kapitoly 4.3. Kapitola 4.4 rozoberá mechanizmus Zero-Copy [18], ktorý predstavuje využitie DMA a mapovanie virtuálnych pamäťových stránok užívateľského priestoru do fyzického priestoru pamäte jadra. V poslednej podkapitole 4.5 sa pozrieme na nový mechanizmus zachytávania paketov PF_RING [20], ktorý výrazne ovplyvnil vývoj na poli zachytávania paketov.

4.1 Eliminácia nadbytočných kópií

Eliminácia nadbytočný kópií je demonštrovaná na aplikácii Webového servera [5]. Ten poukazuje na štyri typy kopírovania:

1. Kopírovanie dát z disku do hlavnej pamäte operačného systému cez I/O operácie. Úsek pamäti predstavuje vyrovnávaciu pamäť pre diskové súbory.

2. Kopírovanie dát z vyrovnávacej pamäte súborov do pamäte aplikácie.

3. Kopírovanie z pamäte aplikácie do pamäte jadra operačného systému.

4. Kopírovanie na pamäť sieťového adaptéra.

Aplikácie typu webový server, často využívajú súbory uložené na disku pre odosielanie klientom. Využívajú pritom I/O operácie, ktoré si často vyžadujúcu spoluúčasť CPU a spotrebovávajú kapacitu I/O zbernice ako aj pamäťovej zbernice. V tomto prípade kopírovanie každého bitového slova dát sú realizované operácie čítania a zápisu. Všetky operácie čítania a zápisu do pamäte spotrebovávajú šírku pásma ovplyvňujúc priepustnosť pamäťovej zbernice a pamäť samotnú. Veľkosť tejto pamäte býva obmedzená aj keď v dnešnej dobe je jej cena nízka čo umožňuje vybaviť systém dostatočne veľkou kapacitou. To sa týka aplikácii bežiacich na klientských staniciach, ktoré typicky neposkytujú žiadne služby narozdiel od serverových aplikácií. U týchto je nutné pracovať s pamäťou efektívne vzhľadom na to, že môžu spracovávať požiadavky veľkého množstva klientov. Pre tento predpoklad uchovávajú často využívané súbory v hlavnej pamäti miesto aby ich zakaždým čítali z disku. Pamäť ako už bolo povedané býva často obmedzená svojou veľkosťou, ktorá u klientských staníc môže predstavovať dostatočne veľký výkon no u servera naopak nedostačujúci stav, ktorý spôsobuje výrazne spomalenie jeho výkonu.

Elimináciu využitia pomalých I/O operácií, u ktorých participuje CPU je možné dosiahnuť využitím DMA. Týmto sa počet operácii redukuje na zápis alebo čítanie [5].

Ako bolo popisované v kapitole 3.1, v pamäťovo mapovanej architektúre sú všetky I/O zariadenia mapované do pamäte. Kopírovanie v bode 3 a 4 preto môžeme zjednotiť za predpokladu, že časť pamäte jadra, konkrétne pamäte pre schránku do ktorej zapisuje aplikácia dáta pre prenos cez rozhranie, je umiestnená práve na sieťovom adaptéri.

Sieťový podsystéme operačného systému však musí pred poslaním dát na rozhranie prepočítať kontrolný súčet. Ten prepočítava v pamäti jadra pre schránku pri zachovaní predošlej architektúry. Ak namapujeme časť pamäte jadra do sieťového adaptéra, zostáva táto operácia buď na CPU pri presúvaní jednotlivých bitových slov alebo na réžií sieťového adaptéra. Prvú týchto metód navrhol pán Van Jacobson a druhú Banks a Prudence [5].

Odstránenie kopírovania z užívateľskej pamäte do pamäte schránky jadra predstavuje riziko. To vzniká tak, že pri prenose aplikačnej oblasti pamäte sieťovým podsystémom môže dôjsť k modifikácii tejto oblasti pamäte aplikáciou čo má za následok nekonzistenciu posielaných dát a spôsobuje chybu pri zápise do schránky [5].

Pre zamedzenie tohto stavu slúži metóda odloženého kopírovania (*Copy on Write,* ďalej COW), ktorá využíva mapovanie virtuálneho adresového priestoru medzi jadrom a aplikáciou. Funguje tak, že pri modifikácii fyzickej stránky vlastníkom detekuje operačný systém stav a prekopíruje stránku vo virtuálnej pamäti pre odkazovanie na pôvodné a modifikované dáta. Týmto zamedzíme samotnému kopírovaniu dát len nastavením niekoľkých deskriptorov.

Mapovanie virtuálny stránok využívajú aplikácie pre urýchlenie kopírovania podobným spôsobom ako u COW. Táto operácia taktiež predstavuje záťaž pri prijímaní veľkého počtu dát zo siete podobne ako obsluha prerušení. Preto bol navrhnutý mechanizmus takzvaných rýchlych vyrovnávacích pamätí (*fast buffers,* ďalej fbufs), ktoré umožňuje predmapovanie virtuálnych stránok na fyzické pre zahájením veľkého prenosu po sieti. Táto operácia môže byť uskutočnená pred začatím prijímania alebo počas prijímania prvých paketov čo spôsobí mierne spomalenie na začiatku.

Rovnako ako metóda COW musí zaistiť aby pri spracovávaní dát nedošlo k ich nekonzistencii použitím príznaku zápisu a modifikácie. U COW je to takzvaný COW bit, ktorý je nastavený jadrom operačného systému v tabuľke stránok u originálnej virtuálnej stránky [5].

Funkcia `mmap()` predstavuje operáciu realizácie mapovania virtuálneho adresového priestoru na fyzický. Je to spoľahlivé systémové volanie umožňujúce aplikáciami uskutočniť spomínané mapovanie.

Ďalšou technikou pre vyhnutie sa zbytočným kopírovaniam naprieč doménou operačného systému je technika vzdialeného priameho prístupu do pamäte (*Remote DMA*). Tá podobne ako klasická operácia DMA vyžaduje informáciu o umiestnení v pamäti, ktorá môže byť prenášaná navrhnutým transportným protokolom.

Všetky kopírovania vo vrstve operačného systému možno redukovať. Týmto sa dostaneme bližšie k potenciálu využitia hardvérovej výbavy a tým aj zrýchlením celého systému [5].

4.2 NAPI

Táto skratka predstavuje nové aplikačné rozhranie (*New API*) pre sieťový podsystém Linux *softnet*. Softnet predstavuje multiprocesorovú-vláknovú architektúru sieťového zásobníku umožňujúceho použiť bežne dostupný hardvér pre realizáciu náročných sieťových aplikácii.

V minulosti predstavoval veľký prichádzajúci prenos problém pri spracovaní vzhľadom k veľkému počtu generovaných prerušení. Linux tak prechádza to stavu zahltenia, bez možnosti odstrániť pakety zo vstupnej rady rozhrania (*backlog*) a tým vykonávať samotné spracovanie paketu. Zavedená hodnota maximálne rýchlosti bez stratovosti paketov pri ich preposielaní (MLFFR) [15] poskytovala základ pre návrh mechanizmov odstraňujúci tento stav. Hodnota MLFFR predstavovala 100% zátaž CPU pri preposielaní paketov [15].

Počiatočné riešenie v tomto smere predstavovali techniky, ktorými ovládače sieťovej karty upozorňovali odosielateľa o zaplnení vstupnej rady následkom ktorého bola znížená prenosová rýchlosť. Vypnutím prerušení pri vypĺňaní vstupnej rady technikou hardvérovej kontroly toku (*hardvér flow control,* ďalej HFC) [16] bol tento problém zmiernený no nie eliminovaný.

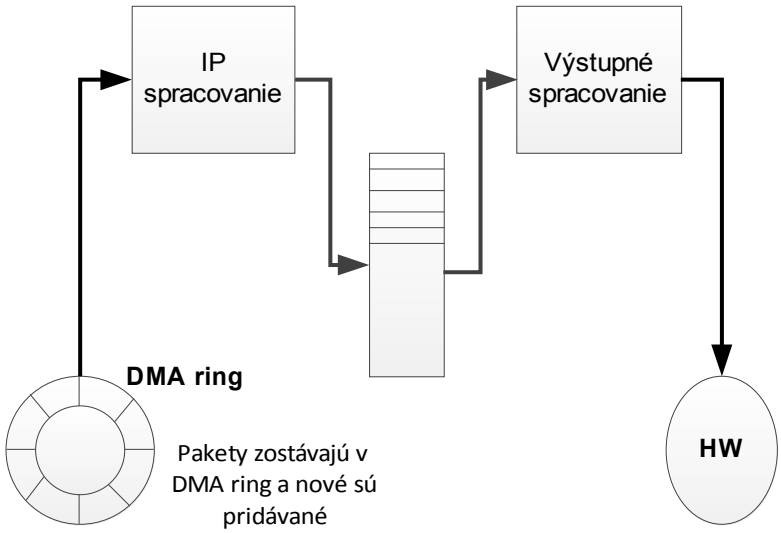
Podobnou technikou náhodného klamstva (*Random lie*) [15] ukazovalo toto riešenie iný pohľad, no prekročením hodnoty MLFFR predstavoval zníženie prenosovej rýchlosti pred dosiahnutím únosnej hranice čo opäť predstavovalo problém.

Ďalším problémom bola zámena poradia paketov, z dôvodu paralelného spracovanie v sieťovom podsystéme IP vláknom v symetrických multiprocesorových systémoch (SMP). SMP predstavuje viac-procesorový systém so zdieľanou pamäťou bežiacim pod jedným operačným systémom.
Pri určení prerušovacej príslušnosti k procesru (IRQ *Affinity*) [15], mohlo jadro procesoru spracovať druhý paket, ktorý dorazil na rozhranie neskor pred prvým.
Všetky tieto riešenia nepredstavovali všeobecne aplikovateľnú techniku, ktorá by splňovala všetky požiadavky na sieťový podsystém pri spracovaní veľkých dátových tokov.

Zavedením NAPI, sa tieto problémy eliminovali. NAPI predstavuje predstavuje hybridné schéma, ktoré využíva metódu `poll()` a zapínanie-vypínanie prerušení. Tým ovplyvňuje vlastnosti systému ako sú latencia (prístupová doba pre získanie obsahu paketov), priepustnosť a zachovanie poradia paketov napríklad v TCP toku.

Pri veľkom dátovom doku dochádza k už spomínanému zahlteniu systému obsluhou prerušení. Pri tomto prenose NAPI vypína prerušenia od sieťovej karty a tým zvyšuje priepustnosť no mierne zvyšuje latenciu k prijatým paketom. Používa tak metódu `poll()` pri neustálom dotazovaní na pamäť prijatých paketov. Pri nízkom dátovom toku to predstavuje spotrebu procesorových cyklov a spomínané zvýšenie latencie. Preto v tomto prípade sú prerušenia opäť povolené a tým sa latencia znižuje.

Celý tento proces sa odohráva bez prítomnosti vstupnej fronty s využitím kruhovej vyrovnávacej pamäte DMA. Tá predstavuje nutnosť zariadení s podporou DMA čo je dnes vo veľkej miere samozrejmosťou. Využitím DMA vytvárame sekvenciu prijatých paketov čím eliminujeme problém poradia v toku.



Obrázok 4.1: NAPI dátová cesta [15]

Takto navrhnuté NAPI umožňuje bezpečne a efektívne využiť možnosti paralelného spracovania v sieťovom zásobníku jadra využitím vlákien.

4.3 Hybridné schémy

Dokument [17] popisuje vytvorenie experimentálnej hybridnej schémy založenej na NAPI a ďalších podobných schémach. Tak ako v prípade NAPI, hybridné schéma predstavuje mechanizmus využívajúci viaceré techniky pri spracovávani prichádzajucého toku do koncového zariadenia.

Výber optimálnej techniky pre zaistenie čo najväčšej efektivity je často náročne vzhľadom k ne-deterministickému prenosu po sieti [17]. Medzi ďalšie spomínané metódy urýchľovania spracovania prenosu podobné NAPI sú:

- Vypínanie a zapínanie prerušení (DE) pri malom a veľkom dátovom toku. Pri veľkom dátovom toku však môže prerušenia spôsobiť viac-krát spomínané zahltenie.
- Zhlukovanie prerušení (*Interrupt Coalescing,* ďalej IC), kde pri danom počte niekoľkých paketov je vyvolané jedno prerušenia namiesto niekoľkých pre každý paket zvlášť.

- NAPI, rozoberané v kapitole 4.2. NAPI využíva zapínanie a vypínanie prerušení spolu s metódou poll.

- Štandardné využitie metódy poll samostatne.

Pri skúmaní niekoľkých parametrov ako priepustnosť, latencia, záťaž na CPU, vykazuje každá metóda iné výsledky. Vo všeobecnosti, najlepšie z toho vychádza metóda DE [17] ale ostatné metódy vykazujú lepšie vlastnosti pri parametroch priepustnosti a latencie.

4.4 Zero-Copy

Ďalšou technikou urýchlenia spracovania paketov je *Zero-Copy* [18]. Táto technika využíva DMA prenos a mapovanie pamäte užívateľského priestoru do pamäte jadra funkciou `mmap()`.

Úlohou Zero-Copy, ako už názov napovedá, je odstrániť kopírovanie, konkrétne medzi sieťovým adaptérom, jadrom a aplikáciou. Vyžaduje modifikáciu ovládača a zaistí tak, že pakety budu kopírované do štruktúry sk_buff vo vybranom mieste pamäte jadra. To zaistí technológia DMA, ktorej ale treba oznámiť kam má kopírovať prijaté dáta. Táto oblasť pamäte je nazývaná paketový priestor [18]. Túto oblasť mapuje do oblasti užívateľského prostredia systémovým volaním `mmap()`. V aplikácii alokujeme následne súvislý priestor virtuálnej pamäti, ktorá je mapovaná do určeného paketového priestoru v jadre. Mapovanie je uložené v tabuľke mapovania fyzických adries a je prístupná sieťovej karte.

Pri mapovaní využíva Zero-Copy techniku COW. Tá výrazne urýchľuje prenos dát v rámci pamäte tým, že prenos je uskutočňovaný len pri zmene originálnych dát čo umožňuje modifikovať len niekoľko deskriptorov. Táto metóda je popísaná v podkapitole 4.1.

Výsledkom metódy Zero-Copy je redukcia kopírovania z priestoru jadra do užívateľského priestoru, ktoré sa odohráva za účasti CPU a systémových volaní. Celý proces je zobrazený na obrázku 4.2.

4.5 PF_RING

Mnoho dnešných aplikácii vykonávajúcich zachytávanie paketov používa štandardnú knižnucu libpcap. Tá ako už bolo spomenutá používa typ schránky PF_PACKET pre ukladanie prijatých paketov, ktoré sú následne spracovávané v štrukturách jadra operačného systému.

Na to by sme urýchlili kopírovanie týchto paketov, musíme použiť schránku, ktorá rovnako poskytuje rozhranie pre ukádlanie paketov do uživateľského prostredia no bez intervencie jadra za účelom schopnosti prijímať pakety na úrovni rýchlosti prenosovej linky. Táto bola navrhnutá v dokumente [20] a nesie názov PF_RING.

Predošlé aplikácie a knižnica libpcap nevykazovali dobré výsledky pri zachytávaní veľkých paketových tokov pohybujúcich sa od sto do niekoľkých stoviek megabitov za sekundu [20].

Pri meraniach uskutočnených sa táto strata pohybovala na úrovni aj deväťdesiatich [20] v závislosti aj na operačnom systéme.

Zavedením techniky *poll*, sa príjem paketov značne urýchlil no stále na nie dostatočnú úroveň, ktorá by mohla odpovedať vysokým prenosovým rýchlostiam. Táto metóda spotrebováva cykly CPU, čo prináša ďalšiu záťaž na systém. Pri volaní metódy `poll()`, ako systémového volania v rámci jadra operačného systému narozdiel od užívateľského prostredia prinieslo ďalší nárast rýchlosti.

Pakety sú narozdiel od kopírovania do štruktúr jadra kopírované do kruhovej vyrovnávacej pamäte, ktorá je vytvorená na začiatku spustenia modulu a umožňuje tak prácu s touto pamäťou. Toto kopírovanie je uskutočňované modulom jadra zviazaným s deskriptorom PF_RING.

Využitím NAPI v spolupráci s novou schránkou PF_RING a modulom jadra, vzniká nový mechanizmus zachytávania pomenovaný podľa nového typu schránky PF_RING. Architektúra celého mechanizmu je na obrázku 4.3.

Okrem štandardných funkcií pre zachytavanie paketov poskytuje knižnica aj využitie spomínaných BPF filtrov a nové typy filtrovania *wildcard* a *precise*. Okrem tejto funkcionality poskytuje aj mechanizmy *balancing* a *clustering*, ktoré umožňujú rozdeliť zátaž spracovania paketov medzi viaceré aplikácie, buď na základe pätice parametrov obsahujúcich zdrojovú a cieľovú IP adresu, zdrojový a cieľový port a aplikačného protokol alebo náhodné rozdelenie metódou *round robin*[1].

Implementácia funkcií komunikujúcich kruhovou pamäťou umožňuje použitím bežne dostupného hardvéru vytvoriť kvalitné zariadenie pre pasívne zachytávanie paketov na úrovni niekoľkých riadkov kódu [20].

Ako bolo spomenuté vyššie, aplikácia komunikuje s kruhovou pamäťou prostredníctvom modulu jadra operačného systému, ktorý mapuje túto oblasť do užívateľského priestoru funkciou `mmap()`. Vzhľadom na to, že oblasť kruhovej pamäti je po spracovaní aplikáciou prepisovaná, je na réžii aplikácii aby tieto pakety ukladala.

Aj napriek tomu, že úmyslom bolo navrhnúť mechanizmus nezávislý na operačnom systéme a použitom hardvéri, využitím niektorých typov sieťových kariet1 možno využiť potenciál knižnice vo väčšom merítku. Ovládače sieťových kariet neboli priamo stavané na zachytavanie paketov ako aj na využitie kruhových pamätí PF_RING, preto býva nutná ich modifikácia. Vzhľadom na to, že nie všetci výrobcovia kariet poskytujú zdrojové kódy ovládačom svojich kariet, je nutné použiť zmieňované.

Okrem týchto ovládačom poskytuje PF_RING aj možnosť využiť ovládače založené na technike Zero-Copy s využitím DMA pre dosiahnutie rýchlosti rovnej rýchlosti prenosovej linky. Tie sú však vydávané pod platenou licenciou, preto je ich použitie obmedzené na päť minút.

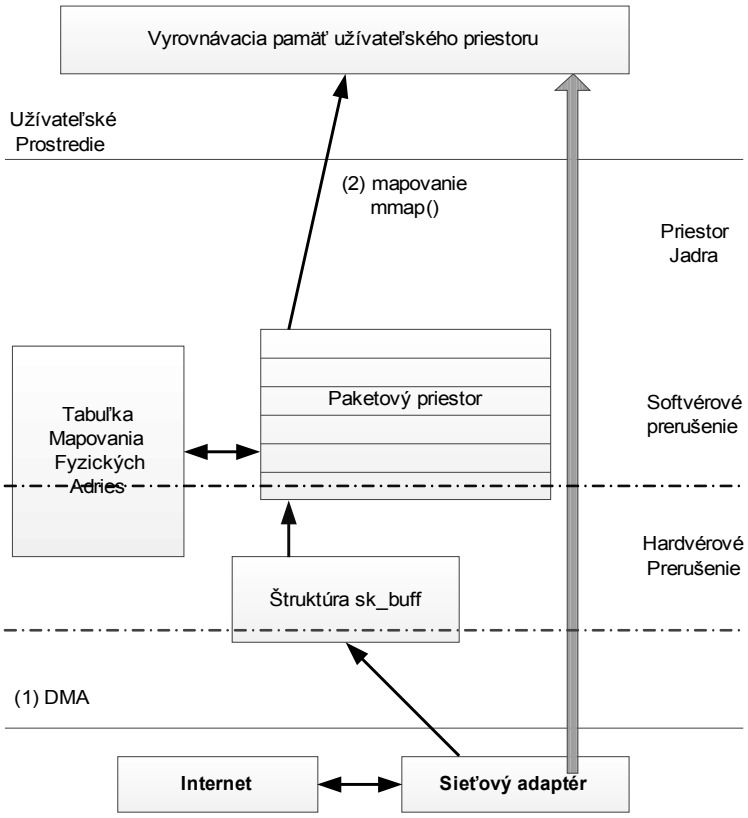
Spomínaný modul jadra pracuje v troch takzvaných transparentných modoch. Prvým mód umožňuje pracovať s bežným hardvérom a využíva NAPI pre dotazovanie sa na prijímané pakety a následne ich presúva do kruhovej pamäte.

Druhý mód využíva potenciál knižnice s využitím špecializovaných ovládačov označených *PF_RING-aware*, a pakety kopíruje súčasne použitím NAPI aj samostatne.

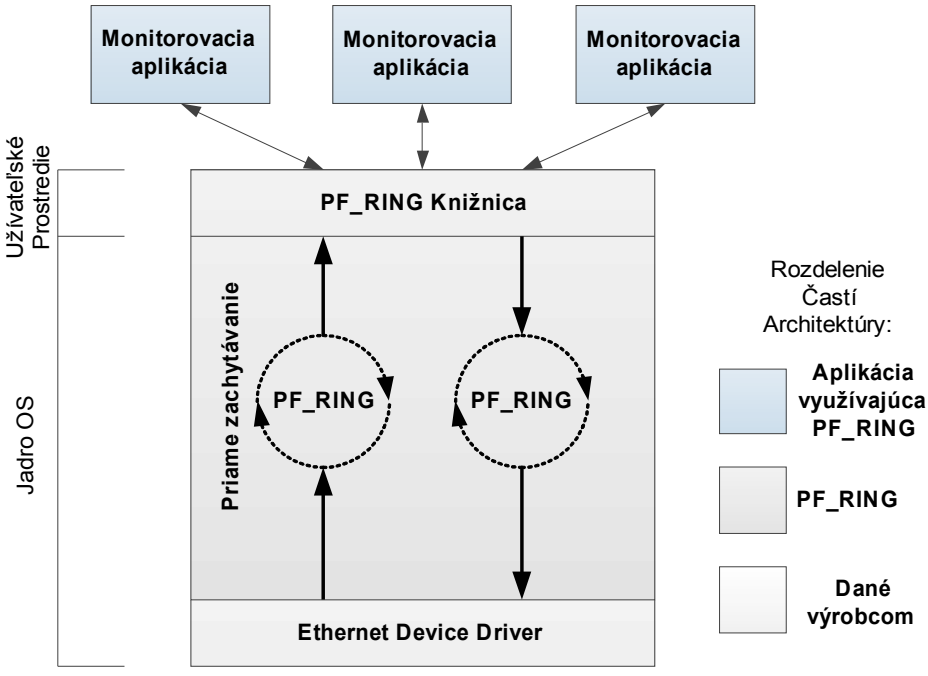
Tretí mód pre dosiahnutie maximálnej rýchlosti vynecháva NAPI kompletne čím zostáva réžia kopírovania paketov do kruhovej pamäte len na ňom.

PF_RING zavádza modifikovanú knižnicu libpcap podporujúcu komunikáciu s kruhovými vyrovnávacími pamäťami PF_RING. Pri použití existujúcich aplikácii je nutné ich znovu preložiť vzhľadom k modifikácie knižnice aby využili potenciál knižnice PF_RING.

1 Informácia obsiahnutá v kruhovej štruktúre pre zamedzenie zmeny poradia a modifikácii prvého prvku.



Obrázok 4.2: Zero-Copy v Linuxe [18]



Obrázok 4.3: Architektúra PF_RING [21]

5 Vývoj aplikácie

Táto kapitola rozoberá návrh a implementáciu nástroja pre odposluch komunikácie v jednotlivých etapách vývoja od analýzy a špecifikácie požiadavkov odvodených z teoretického základu v predošlých kapitol 2 a 3. Kapitola 5.4 rozoberá konkrétny návrh, z ktorého potom vychádza implementácie v kapitole 5.5. Kapitola 5.6 profiluje implementované riešenie.

5.1 Softvérové a hardvérové vybavenie

Aplikácia je napísaná v jazyku C. Jedná sa o procedurálny programovací jazyk umožňujúci dekompozíciu problému v návrhovom zhora nadol. Pri implementácii sme využili grafické vývojové prostredie Code Blocks a štandardný Linuxový textový editor Vim. Projekt bol prekládaná na prekladači gcc vo verzii 4.4.5.

Použitím knižnice PF_RING a využitie jej potenciálu bolo nutné zakúpiť sieťový adaptér od firmy Intel [23], vzhľadom k nutnej modifikácie ovládačov. Tie sú dostupné so zdrojovými súbormi knižnice PF_RING, ktorá je vydávaná pod licenciou GNU GPL a umožňujú dosiahnuť nárast výkonu pri zachytávaní paketov oproti štandardnému *Vanilla* PF_RING. Počítač desktopové typu realizujúcu sieťový prvok s podporou odposluchu obsahuje dve sieťové karty. Jednu pre prepojenie so server disponujúcou rýchlosťou 100 megabitov za sekundu a druhú už spomínanú kartu pre samotný odposluch od firmy Intel.

Všetky počítače navrhovanej architektúry pre testovanie a implementáciu využívali operačný systém Debian[1] GNU/Linux vo verzii 6.0.4 s kódovým označením *Squeeze* a súčasnou verziou jadra 2.6-32-5-amd64.

Na testovacie účelý a simuláciu reálneho prostredia pri nasadení odposluchu sme využívali hardverový generátor prenosu od firmy Spirent s označením SPT-2000A a softvérový generátor Ostinato [22]. Obe tieto generátory umožňujú široké možnosti nastavenia generovaných rámcov, v rôznych veľkostiach, rýchlostiach a protokolovej sade.

5.2 Špecifikácia požiadavkov

Požiadavky na navrhovanú aplikáciu vychádzajú z referenčného modelu pre zákonné odposluchy v kapitole 2. Predstavujú aplikáciu, schopnú filtrovať a zachytávať komunikáciu od subjektu určeného pre odposluch. Odposluch prebieha počas komunikácie určeného subjektu na základe pridávaných či odoberaných filtrovacích pravidiel. Samotný nástroj na odposluch alebo sonda, musí byť ovládaná vzdialene, príkazmi pre pridanie alebo odobranie filtrovacích pravidiel. Musí byť známe jej umiestnenie v sieťovej infraštruktúre. Pre kontrolu aplikácie pred vykonávaním samotného odposluchu je nutné vykonať autentizáciu zo strany ovládacieho centra.

Vzhľadom na to, že komunikácia subjektu môže byť zastúpená vo veľkom dátovom toku je nutné zabezpečiť aby nedošlo k strate vyžiadanej informácie a zaistiť spracovanie len vybraných paketov. Tie sú poslané na určenú destináciu špecifikovaným rozhraním rozdielnym od toho, na ktorom prebieha odposluch.

Získané dáta musia byť zálohované pre ďalšiu analýzu v človeku rozpoznateľnom formáte. Pre rozpoznanie protokolov musia byť známe ich hlavičky aby bolo možné uskutočniť preklad z formátu získaného pri prenose. To má za úlohu funkčný blok referenčného modelu MF.

1 *htttp://www.debian.org*

5.3 Analýza požiadavkov

Na základe požiadaviek z predošlej kapitoly, sme sa rozhodli pristupiť k bližšej špecifikácii jednotlivých komponent celého systému a častí aplikácie.

Aplikácia predstavuje aktívny paketový zachytávač, nazvime ho klient, ktorý preposiela získané informácie na vopred stanovenú destináciu. V referenčnom modeli z kapitoly 2 má zastúpenie v zariadení CC-IIF. Destinácia predstavuje aplikáciu, ktorá kontroluje klienta a generuje informáciu súvisiacu s odposluchom. Táto časť je v referenčnom modeli obsiahnutá v zariadení IRI-IIF a CCTF. Ak tieto zariadenia spojíme, môžeme predpokladať vytváranie a zároveň preposielanie požiadaviek na odposluch v jednej aplikácii, nazvime ju server. Tá kontroluje klienta a overuje jeho identitu autentizačným procesom. Jeho umiestnenie musí vedieť vopred, zvyčajne od AF podľa referenčného modelu, no my ju môžeme nahradiť statickým nastavením. Klient je ovládaný prostredníctvom CCCI rozhrania.

Požiadavka na odposluch obsahuje identifikátor odposluch, v našom prípade ho označíme SID (*Session Identification*). V referenčnom modeli predstavuje identifikátor LIID. Ďalej obsahuje špecifikáciu filtrovacích pravidiel CC, ktorá pozostáva z IP adresy, časového údaju o začiatku a koncu odposluchu. Tieto informácie sú prenášané rozhraním CCCI v textom formáte správ. Identifikátorom označené dáta potom posielame v dopredu stanovenom formáte prenosového protokolu cez rozhranie INI3.

Pre zachytávanie paketov použijeme techniku PF_RING obsiahnutú v kapitole 4.5. Výber tejto techniky vychádza z predpokladu dosiahnutia najvyššej efektivity s použitím špecializovaného hardvéru sieťovej karty a jej ovládača.

Získane dáta cez rozhranie INI3 zapisujeme do súboru.

5.4 Návrh

Spomínaným častiam referenčného modelu venujeme pozornosť vo fáze návrhu. Vytváraná aplikácia je v reálnom svete súčasťou sieťovej infraštruktúry pozostávajúcej z viacerých koncových bodov a aktívnych sieťových prvkov. Preto pre zjednodušenie naša navrhovaná architektúra predstavuje tri počítače pre zastúpenie všetkých účastníkov systému na odposluch komunikácie:

1. Subjekt určený pre odposluch.
2. Nástroj, ktorý zachytáva vyžiadanú komunikáciu, pomenovaný klient.
3. Server, ktorý uchováva zachytené informáciu od subjektu, získanú od klienta.

Vzhľadom k tomu, že subjekt je v reálnom svete súčasťou väčšej množiny koncových bodov, predpokladáme využite generátorov paketových prenosov simulujúcich toto prostredie.

Nástroj, samotné jadro architektúry, je zastúpený desktopovým počítačom, ktorý obsahuje viaceré sieťové rozhrania umožňujúce na jednej strane zachytávať a na druhej preposielať pakety na server. Server využíva pre záznam zachytených paketov súbor na disku.

Jednotlivé súčasti architektúry ako aj navrhované programy budú popísané v ďalších podkapitolách.

5.4.1 Odposluchávaný subjekt

Ako bolo popisované v predošlej podkapitole 5.4, subjekt pre odposluch býva súčasťou rozsiahleho sieťové prostredia s predpokladom veľkých prenosov. Simulácia prostredníctvom generátorov prenosu nám umožňuje priblížiť sa reálnemu prostrediu, v ktorom je samotný odposluch vykonávaný.

Pre použitie v našom systéme a overenia správnosti fungovania nám postačuje generovanie prenosu na úrovni niekoľkých desiatok až stoviek megabitov za sekundu. To môžeme docieliť použitím niektorého zo širokej škály nástrojov dostupných na internete v neplatenej licencii. Vo fáze návrhu bol použitý softvérový generátor *Ostinato* [22]. Pri testovani sme sa operiali aj o výsledky dosiahnuté využitím hardvérového generátora Spirent SPT-2000A.

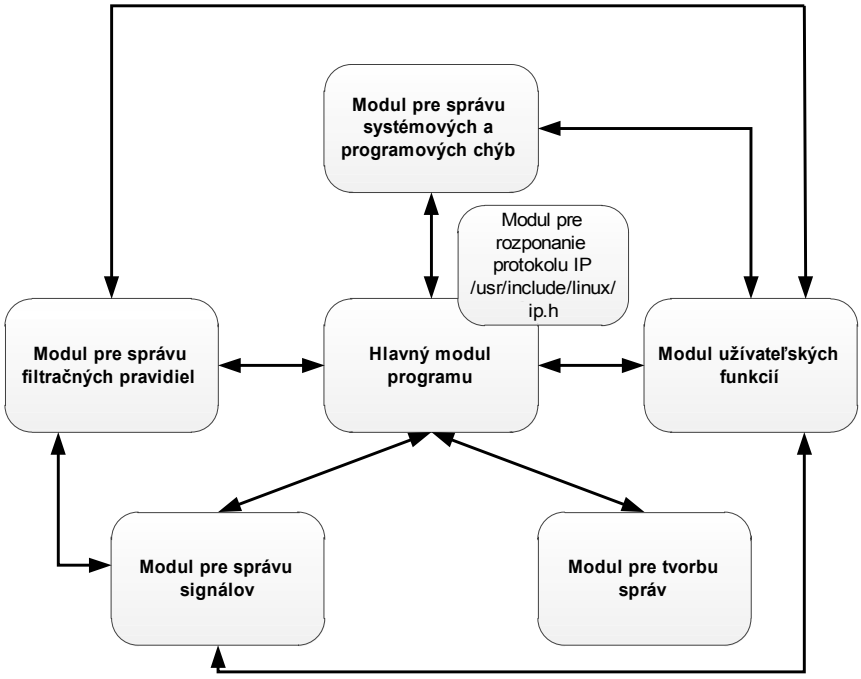
5.4.2 Klient

Aplikácia klienta tvorí základ celej architektúry pre odposluch obsahu komunikácie. Jeho návrh, vychádza z dekompozície problému na programové moduly zastávajúce jednotlivé podúlohy. Názvy modulov zobrazených na obrázku 5.1 reflektujú ich funkcie vo všeobecnosti.

Jadrom celého návrhu je *hlavný programový modul* vykonávajúci niekoľko funkcií. Pred samotným započatím zachytávania je nutné nadviazať komunikáciu so serverom. Ten, ako už bolo spomenuté vo fáze špecifikácie a analýzy požiadavok, komunikuje s klientom a kontroluje jeho činnosť. Jeho návrh a podobná dekompozícia problému je popisovaná v nasledujúcej podkapitole 5.4.3.

Prvým bodom návrhu bolo vytvorenie riadiaceho kanálu CCCI pre spustenie autentizačného procesu klienta a výmenu správ špecifikujúcich pridávanie či odoberanie filtrovacích pravidiel. Druhým bodom bolo vytvorenie kanálu INI3 pre preposielanie zachytených dát na server.

Kanál CCCI je používaný už pre spomínané spustenie autentizačného procesu a zároveň prijíma a odosiela správy od serveru. Narozdiel od CCCI, kanál INI3 je využívaný len na preposielanie odfiltrovaných správ.



Obrázok 5.1: Dekompozícia aplikácie klienta na moduly

Vzhľadom na to, že spomínané kanály majú odlišnú funkciu a určený smer komunikácie, bola navrhnutá schéma podobná protokolu FTP [24], ktorý využíva riadiaci kanál pre kontrolu prenosu a komunikačný kanál pre dáta. Obe kanály CCCI aj INI3 využívajú TCP spojenie, ktoré zaisťuje kontrolu prenosu medzi klientom a serverom.

Zahájením hlavného vlákna procesu klienta, sa vytvorí spojenie s bežiacim serverom a klient odošle autentizačný reťazec. Ten má nasledujúci formát:

```
CC probe_klient xzimas00\n\0
```
(5.1)

Tento reťazec obsahuje informáciu o názve aplikácie klienta a autentizačný reťazec v podobe prihlasovacieho mena. Pokračuje oddeľovacím znakom konca riadku používanom v Linuxových systémoch. Nakoniec obsahuje ukončovací znak nuly pre dodržanie konvencie formátu reťazca v programovacom jazyku C. Táto správa je poslaná na server pre overenie, ktoré je súčasťou popisu v kapitole 5.4.3.

Po odoslaní reťazca klient vytvorí nové vlákno pre riadiaci kanál CCCI a kanál INI3. Po overení klienta na strane servera dôjde buď k zlyhaniu spojenia na oboch kanáloch v prípade negatívneho výsledku autentizácie alebo začne cyklus prijímania nových požiadaviek na odposluch v novom vlákne vytvorenom pre CCCI správy.

Pre správy vymieňané prostredníctvom kontrolného rozhrania CCCI bol navrhnutý textový prenosový protokol. Ten predpisuje formát správ pre pridávanie či odoberanie filtrovacích pravidiel a odpovedí na tieto správy.

Všetky správy vychádzajú zo všeobecného formátu:

```
(operácia, "cc-iif", parametre)\n\0
```
(5.2)

Reťazec `operácia` definuje či bude pravidlo špecifikované parametrami v časti `parametre` pridávané alebo odoberané. Pre pridávanie pravidla má správa nasledujúci formát:

```
('new_intercept', 'cc-iif', SID, NID, start odposluchu, koniec
                                odposluchu)\n
```
(5.3)

a pre odoberanie pravidla:

```
('delete_intercept', 'cc-iif', SID)\n\0
```
(5.4)

Na obe tieto správy reaguje klient odpoveďou v podobe:

```
('ack', povodna sprava)\n\0
```
(5.5)

Parameter *SID* (*Session ID*), jednoznačne identifikuje odposluch, a predstavuje celočíselný údaj. Parametrom *NID* (*Network ID*), identifikujeme subjekt, určený pre odposluch vo formáte reťazca čísel oddelených bodkami pre IPv4 formát adresy (*Dot-Decimal Notaion*). Poslednou dvojicou parametrov je čas pre začiatok a koniec odposluchu. Tie sú reprezentované celočíselným údajom v počte sekúnd od začiatku éry Unixu v roku 1970.

Takto definované pravidlá sú uložené v abstraktnej dátovej štruktúre obojsmerne viazaného zoznamu. Klient ich po spracovaní pridáva a odoberá využitím funkcií v *module pre správu filtračných pravidiel.* Zároveň pre odstránenie pravidla využíva parameter SID obsiahnutom v správe pre operáciu *delete_intercept.*

Odpoveď na prijaté správy je vytváraná v hlavnom programovom module vo vlákne pre príjem CCCI správ.

Krátko po otvorení INI3 kanálu pre posielanie zachytených dát, vzniká samotný proces zachytávania na vopred definovanom rozhraní.

Pre toto rozhranie je definovaný PF_RING deskriptor, ktorý reaguje na prijaté správy volaním funkcie na ich spracovanie.

Vzhľadom na to, že obe spojenia CCCI aj INI3 pracujú nezávisle od seba, je nutné ich nejakým spôsobom synchronizovať pre kontrolu filtrovacích pravidiel.

Synchronizácia spočíva v prístupe do zdieľanej pamäte, ktorá predstavuje filtrovacie pravidlá uložené v abstraktnej dátovej štruktúre zoznamu. Tá je definovaná v hlavičkovom súbore modulu pre správu filtračných pravidiel. Prijatím správy od na CCCI rozhranie sa spustí proces kontroly a editácie tejto dátovej štruktúry.

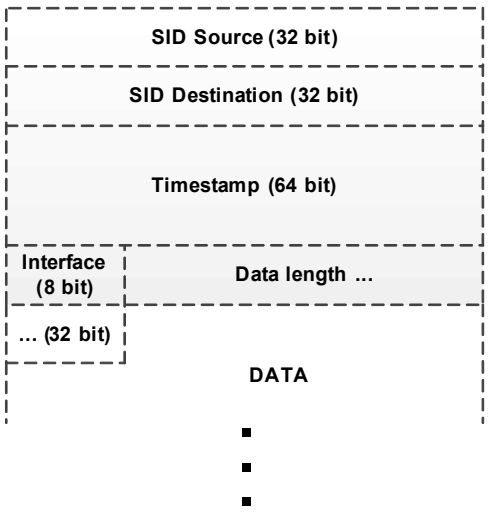
Po predaní alebo odobraní pravidla povolí vlákno, ktoré prijíma CCCI správy prístup k tejto štruktúre funkcii vykonávajúcej spracovanie paketov v hlavnom programovom vlákne.

Pre kontrolu paketu je nutné rozpoznať prenášaný protokol obsahujúci informáciu na základe ktorej je filtrovaný. Pre rozpoznanie prenášaných protokolov je možné využiť knižnicu PF_RING vytvorením zásuvného modulu (PF_RING *plugin*). Ten je následne vyhodnocovaný v module jadra s aplikovaním filtrovacieho pravidla definovaného ako parameter jeho štruktúry. Tieto moduly sú prevažne využívané pre tvorbu štatistik, no nám postačuje informácia v podobe hlavičky protokolu IPv4 definovaná v súbore */usr/include/linux/ip.h*.

Získaním informácie o IPv4 adrese z hlavičky paketu následne realizujeme kontrolu v zozname pravidiel funkciou definovanou v *module užívateľských funkcií.*

Po následnej zhode niektorého z pravidiel s adresou obsiahnutou v hlavičke prijatého paketu vykonávame tvorbu správy v *module pre tvorbu správ.* Vytvorenú správu posielame cez schránku vytvorenú pre INI3 kanál na server.

Podobne ako u CCCI rozhrania, pre správy posielané na server INI3 rozhraním používame predpísaný formát definovaný navrhnutým protokolom. Hlavička tohto protokolu je na obrázku 5.2.



Obrázok 5.2: hlavička protokolu definovaného pre prenos správ cez INI3 rozhranie

V prípade, že paket vyhovuje pravidlu v zozname je nutné identifikovať umiestnenie IPv4 adresy v rámci hlavičky pre vytvorenie INI3 správy s predpísanou hlavičkou. V tej je identifikované, či IPv4 adresa paketu bola na mieste zdrojovej alebo cieľovej časti.

Ak IPv4 adresa odkazuje na zdrojovú adresu, novo vytvorená správa s INI3 hlavičkou obsahuje ma mieste identifikátora *SID Source*, identifikátor SID vyhovujúceho pravidla v zozname. V prípade, cieľovej IPv4 adresy je nastavený identifikátor *SID Destination*.

Informácia získaná z hlavičky paketu tentokrát definovanej knižnicou PF_RING `struct pfring_pkthdr`, obsahujúca čas príchodu paketu na odposluchávacie rozhranie, je zapísaná v rovnakom formáte do časti INI3 hlavičky *Timestamp.* Toto rozhranie je identifikované číslom,

napríklad 1 pre rozhranie `eth1` v časti *Interface* INI3 hlavičky. Posledná informácia v INI3 hlavičke popisuje dĺžku dát samotného paketu získanej z jeho hlavičky definovanej knižnicou PF_RING. V poslednej fáze je pripojený celý zachytený paket predstavujúci dáta komunikácie CC. Opäť pre použitie TCP toku a schránok je nutné ukončiť tú správu sekvenciou dvoch znakov konca riadku v Linuxe.

Správy sú v takomto formáte posielané na server, ktorý ich spracováva. Celé spracovanie a posielanie možno popísať obrázkom 5.3.

Obrázok 5.3: Proces zachytávani, filtrovania a posielania paketov u klienta

Pre maximálne využitie potenciálu knižnice PR_RING využívame špecializovanú sieťovú kartu [23]. Tá spolu s modifikovanými ovládačmi dostupnými s knižnicou umožňuje efektívnejšie kopírovanie paketov do kruhových pamätí a tým zvyšujú celkovú priepustnosť systému.

Nastavením obsluhy signálov `SIGUSR1` a `SIGUSR2` sme docielili možnosť vypísať aktuálne uložené pravidlá zo serveru ako aj zobraziť štatistiky prijatých, zahodených a vyfiltrovaných paketov.

Programy klient aj server bežia v nekonečnom cykle vykonávajúc svoje operáci a pre ukončenie je nutné vykonať príkaz `Ctrl+C`, ktorý spôsobí vyvolanie signálu `SIGINT`. Obsluha toho signálu spúšťa ukončovaciu fázu a uvoľňovanie pamätí alokovaných zdrojov.

Posledným článkom návrhu je *modul pre správu systémových a programových chýb*, ktorý poskytuje funkcie pre výpis chybových hlásení na štandardný chybový výstup. Použitím vlákien je nutné uchovávať premennú `errno`, nastavovanú pri zlyhaní niektorého zo systémových volaní vzhľadom k tomu že obe vytvorené vlákna využívajú rovnaký užívateľský adresový priestor. Tento modul je využívaný pri všetkých operáciach klienta.

5.4.3 Server

Štruktúra návrhu aplikácie servera je na obrázku 5.4. Opäť jadro zahrňuje *hlavný programový modul*.

Pri spustení podobne ako klient vytvorí komunikačné kanáli CCCI a INI3. Rozdiel oproti klientovi predstavuje vytvorenie samotného vlákna pre rozhranie INI3, pre zahájením autentizačného procesu.

Ten prebieha tak, že server vytvorí naslúchaciu schránku, ktorá čaká na spojenie od klienta. Ten sa autentizuje poslaním reťazca (5.1). Následne *modul pre autentizáciu* zavolá funkciu pre overenie identity klienta. Používa pri tom porovnanie zadaného reťazca a reťazca obsiahnutého v správe od klienta (5.1).

Pri zhode vypíše úvodné okno a očakáva vstup pre zadávanie príkazov na odposluch. Ten sa dotazuje na vytvorenie alebo mazanie, podľa protokolu definovaného v predošlej podkapitole. Pri vytváraní, získava všetky informácie od zadávateľa, ktoré konvertuje do formátu predpísaného protokolu v *module pre tvorbu správ*. Vyžaduje vyplnenie všetkých polí definujúcich informácie súvisiace s odposluchom a to:

1. SID číslo, ktoré identifikuje odposluch.

2. IPv4 adresu v spomínanom formáte reťazca oddeleného bodkami

3. Nasledujúce informácie o začiatku a konci odposluchu:
   1. Rok.
   2. Mesiac.

3. Deň.
4. Hodinu.
5. Minúty.
6. Sekundy.

Po vytvorení kontroluje správnosť zadaných údajov a preposiela správu na rozhranie CCCI pre doručenie klientovi. Po odoslaní správy čaká na potvrdenie vo formáte (5.5). Prijatím správy potvrdzujúcej pridanie alebo odobranie pravidla zo zoznamu na strane klienta sa celý cyklus opakuje. Potvrdzujúce správy sa rovnako kontrolujú pre dodržanie predpisu správ daného navrhnutým protokolom podľa obrázka 5.2. Túto funkciu zaisťuje *modul užívateľských funkcií.*

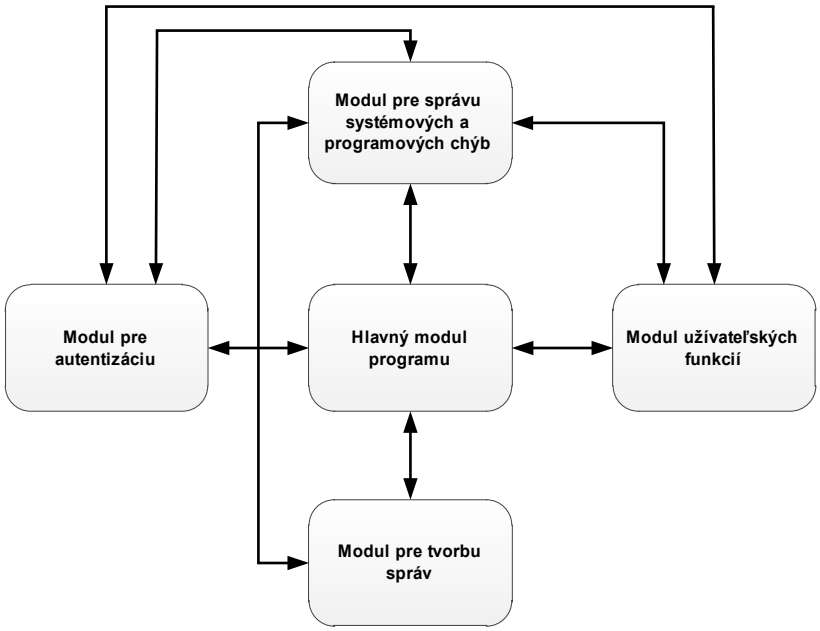
Mazaním pravidla zadávame len identifikátor odposluchu SID, ktorým sa spomínaný odposluch na strane klienta odstráni zo zoznamu.

Hlavnou časťou je realizácia pasívneho odposluchu ukladaním prijatých dát od klienta prostredníctvom rozhrania INI3. Na základe teórie operačných systémov v kapitole 3.2, sú pomalé I/O operácie zápisu dát do súboru nahradené kopírovaním prijatých dát do vyrovnávacej pamäte v našom prípade predstavujúcej pole reťazca znakov. Kontrolou pretečenia tohto poľa je iniciovaní zápis obsahu na pevný disk. Operácia je vykonávaná v module užívateľských funkcií.

Úplný zápis do súboru je vykonaný až po uzatvorení spojenia INI3 iniciovaného klientom. To z dôvodu použitia blokujúcej funkcie, ktorá aktívne čaká na ďalšie dáta so súčasnými uloženými v prijímacej pamäti statického poľa. Po ukončení spojenia, dôjde k termínácii vlákna pre INI3 rozhranie.

Obe vlákna, hlavné CCCI a vytvorené INI3, pracúju rovnako ako klient v nekonečnom cykle preto je nutné ukončenie behu programom využitím klávesovej skratky `Ctrl+C` a vyvolaním signálu `SIGINT`, ktorého základnou obsluhou je ukončenie behu programu.

Keďže server má reprezentovať zariadenie, ktoré vytvára záznamy o odposluchu v zrozumiteľnom formáte bežného jazyka bol navrhnutý samostatný program vykonávajúci túto funkciu z dôvodu urýchlenia spracovania na strane servera. Táto konverzia z hexadecimálneho formátu do formátu bežného jazyka predstavuje I/O operácie s diskom, tak by predstavovala nemalé opozdenie na strane servera. To už k existujúcim podmienkam využitia desatiny rýchlosti sieťovej karty na ktorej prijíma INI3 správy prichádza neprípustné.



Obrázok 5.4: Dekompozícia problému servera na moduly

5.5 Implementácia

V tejto kapitole rozoberieme implementácia jednotlivých účástníkov narvhnutej architektúry z podkapitoly 5.4.

Prvou implementovanú súčasťou bol server. Ten musí byť spustený pred samotnou realizáciou odposluchu keďže sčasti predstavuje zariadenie CCTF, ktoré ovláda sondu pre odposluch.

Adresová a portová konfigurácia je staticky nastavená v hlavičkovom súbore `probe_server_main.h`. Preto je nutné pre zmenu nastavenia editovať tento súbor. Konfigurácia je uložená v štyroch konštantách definovaných:

```
1. #define SERVER_CCCI_ADDRESS "192.168.0.1"
2. #define SERVER_INI3_ADDRESS "192.168.0.1"
3. #define SERVER_CCCI_PORT 6000
4. #define SERVER_INI3_PORT 6001
```

Pred samotným ustanovením spojenia vyžítím schránok BSD je nutné inicializovať adresové štruktúry. Tie sú definované v súbore `unistd.h` systému linux. Predstavujú štruktúru typu `sockaddr_in`, ktorá je použitá pri vytváraní schránky funkciou `socket()` definovanou v súbore `sys/socket.h`.

Po vytvorení vlákna využitím funkcie zo súboru `pthread.h` bolo nutné linkovať knižnicu `lpthread` s naším programom.

Pred vytvorením vlákna pre INI3 rozhranie, inicializujeme globálne adresové štruktúry `server_ccci` a `server_ini3`. Tie použijeme pri vytvorení súborového deskriptoru volaním funkcie `socket()`.

Každý z vytvorených deskriptorom použijeme na vytvorenie naslúchacích a pripojených deskriptorov. Tie vytvoríme sekvenciou funkcií `bind()`, `listen()` a `accept()`. Funkciou `bind()` naviažeme adresnú a portovú konfiguráciu s predávaným deskriptorom. Následne vytvoríme deskriptor, naslúchajúci na danej adrese pre pripojenie klienta. Ten vytvorí podobnou sekvenciou funkcií `socket()` a `connect()` spojenie so serverom na rozhraní CCCI a INI3. Po pripojení funkciou `accept()` na strane servera zahajíme komunikáciu na oboch kanáloch.

Nasleduje autentizačná fáza iniciovaná klientom. Ten zapíše funkciou `write()` na pripojený deskriptor reťazec (5.1). Po prijatí správy serverom blokujúcou funkciou `read()` do reťazca `received_ccci_line`. Po zistení konca správy identifikovanom znakom nového riadku `\n` a ukončovacím znakom `\0` ukončíme cyklus čakania na ďalšie správy.

Načítaný reťazec posunieme ako parameter funkcii `authentication_phase()`. Tá s využítím regulárnych výrazov a funkcíi zo súboru `regex.h` overí správnosť formátu. Z potvrdeného výsledku vezme reťazec obsahujúci prihlasovacie meno a zahájí načítanie reťazca z klávesnice. Pri zhode zadaného prihlasovacieho mena z klávesnice a toho obsiahnutého v správe vráti funkcia hodnotu `0` pre úspešnú autentizáciu.

Po úspešnej autentizácii zobrazí úvítaciu správu so súčasnou konfiguráciou a spusti cyklus vytvárania požiadavkov na odposluch. Ten opakuje sekvenciu vytvárania reťazca správy podľa predpísaného formátu prokolu, funkciami `create_message()` a funkciami pre pridanie `new_intercept()` a odobranie `delete_intercept()` odposluchu zo zoznamu klienta.

Obe tieto funkcie pracujú s regulárnymi výrazmi pre overenie správnosti zadávaných údajov z klávesnice a po vykonaní úspešne skončenej kontroly alokový reťazec obsahujúci správu vrátia funckii `create_message()` a tá ho vráti do hlavného vlákna programu servera. Ten prepošle reťazec funkciou `write()` prostredníctvom pripojeného deskriptora CCCI rozhrania.

Nasleduje fáza spracovania požiadavku na strane klienta a odpoveď vo formáte (5.5). Súčasne s vytvorením komunikačných kanálov CCCI a INI3 na strane klienta s už spomínanými funkciami `socket()`, `connet()` a inicializáciou adresových štruktúr globálne nadefinovaných v súbore

probe_client_main.c vzniká PF_RING deskriptor pre ovládanie kruhových pamätí prostredníctvom modulu jadra. Ten je vrátený funkction pfring_open(), ktorej parametrami sú podobne ako u využitia knižnice libpcap a jej funkcie pcap_open_live() názov rozhrania (v našom prípade rovnako staticky definovanom v súbore probe_client_main.h konštantou CAPTURE_DEVICE), maximálna veľkosť prenášaného paketu (pre uspokejenie všetkých potrieb sme zvolili maximálnu veľkosť 65535 bajtov) a nakoniec mód karty (PF_RING_PROMISC).

Voľba módu PF_RING_PROMICS je kľúčová, inač by nám rozhranie pakety zahadzovalo z dôvodu, že neboli určené pre neho. Vrátený deskriptor použijeme u funkcie pcap_loop(), ktorou spustíme zachytávanie a spracovanie paketov.

Jej parametrom je odkaz na funkciu, ktorá samotné spracovávani vykonáva. Ta beží v hlavnom programe a má názov process_packet.

Na chvíľu sa vrátime k prijímaniu CCCI správ na základe ktorých prebieha spracovanie paketu. Pri prijatí CCCI správy klientom funkciou read(), hľadáme ukončovací reťazec určujúci koniec správy. Po jeho nájdení funkcou ststr() zo súboru string.h zahájime fázu pridávania alebo odoberania odposluchu.

Dátová štruktúra zoznamu pravidiel na odposluch je definovaná v súbore probe_client_interception.h a má nasledujúcí tvar:

```
typedef struct interception {
 struct interception * previous;
 struct interception * next;
 int * sid;
 struct in_addr * ipv4_address;
 time_t * start;
 time_t * stop;
} *P_INTERCEPTION;

typedef struct list_intrceptions {
 P_INTERCEPTION active_element;
 P_INTERCEPTION first_element;
} TLIST_INTERCEPTION;
```

Tieto nadefinované štruktúry využívame v celom programe klienta. Predtým však, musí byť zoznam zoznam inicializovaný. To sa deje funkciou Init() na začiatku vykonávania celého programu.

Po inicializácii môžeme prijatú správu obsahujúcu reťazec příkazu použiť funkciou edit_interceptions() a vykonať danú operáciu. Táto funkcia rovnako ako v predošlých prípadoch využíva knižnicu pre regulárne výrazy vložením súboru regex.h. Regulárnym výrazom opäť kontrolujeme správnosť formátu prijatých správ (5.3) a (5.4). Po overení správnosti výrazu v správe aplikujeme operáciu na základe jej prvej časti. Vo výsledku je pridanie alebo odobranie pravidla funkciami InsertFirst() alebo DeleteInterception().

Po vložení je vo vlákne CCCI na strane klienta vytvorená potvrdzujúca správa ktorú očakáva server pre ďalšie pokračovanie vo svojej činnosti. Tento interval nie je nijak obmedzený.

Ak sa vrátime k spracovávania, prídeme k ďalšiemu spornému bodu implementácie ktorý bol vyriešený zamykaním a odomykaním prístupu k zdieľanej pamäti reprezentovanej zoznamom pravidiel.

Pri prijatí paketu prevedieme jeho reťazec predstavujúci pole znakov typu unsigned char na štruktúru hlavičky IP protokolu. Tá predstavuje v zjednodušení zásuvny modul v podobe súboru ip.h. Touto konverziou získame informácie na porovnávanie. Po uzamknutí pristupu pre CCCI vlákno voláme funkciu packet_check(). Kontrolou jej návratového kódu určíme ďalšie spracovávanie paketua na základe príznaku predávaného do funkcie odkazom určíme, či bola zhoda na strane zdrojovej alebo cieľovej IP adresy.

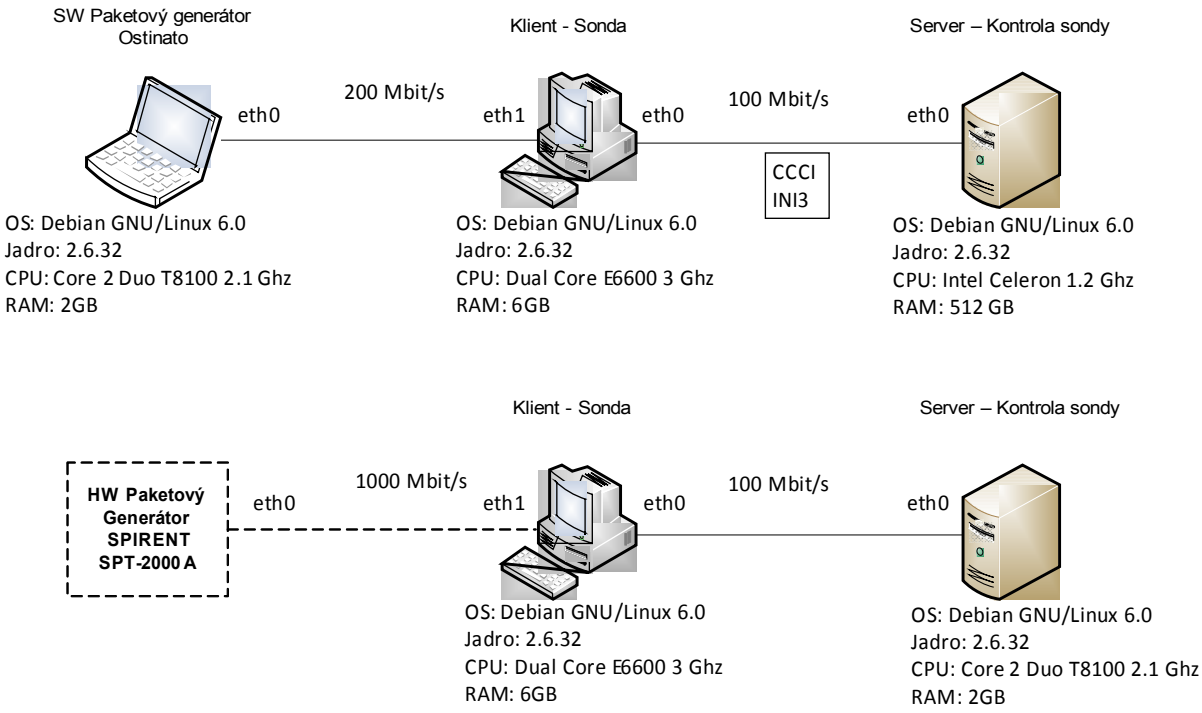
Pri návratovom kóde 0 budeme vytvárať formát spravy pre posielanie rozhraním INI3. Správa je vytvárana modulom `probe_client_msg.c`. Po jej vytvorení prepošleme správu funkciou `write()`.

Na strane servera nastáva spracovanie tejto správy. Prijatím funkciou `read()` vkladáme správu do staticky definované poľa znakov pre urýchlenie operácii vzhľadom na ukladanie do súboru, ktoré by predstavovalo vyššiu časovu náročnosť. Veľkosť tohto poľa určuje konštanta `MAX_INI3_MSG_BUFFER_SIZE` a je nastavená na hodnotu `1000000` v hlavičkovom súbore `probe_server_main.h`.

Prostredníctvom funkcie `content_of_communication_processing()` ukladáme získaného dáta do tohto poľa a pri jeho zaplnení do súboru. Kontrolu aktuálnej veľkosti uskutočňujeme staticky definovanou premennou vo vnútri funkcie, čo ma za následok jej umiestnenie v statickej oblasti pamäte programu a tým zachovanie hodnoty medzi jednotlivými volaniami.

Ako bolo spomenuté vo fáze návrhu, kompletné dáta sú získané až po ukončení spojenia cez kanál INI3. Tým sa zapíše zostavajúci obsah poľa do súboru.

Po ukončení programu servera je možné spustiť program `convert`, ktorý vykoná časovo náročnú operáciu formátovania hexadecimálnych znakov uložených v prvom súbore zachytených paketov a vytvorí súbor s názvom `content_of_communication.readable` pre možnosť kontroly odchytených paketov.



Obrázok 5.5: Navrhnutá a testovaná architektúra

5.6 Profilovanie implementovaného riešenia

Implementovaná aplikácia používa návrh jednoduchého filtrovacieho pravidla obsiahnutého v CCCI správe. Pri návrh zložitejších pravidiel, napríklad využitím pätice definujúcej zdrojovú a cieľovú IP adresu, zdrojový a cieľový port transportného protokolu spolu s informáciou o prenášanom protokole vyššej vrstvy je možné dosiahnuť problém s náročnou kontrolou pravidla v rámci dátovej štruktúry zoznamu.

Pre tento typ filtrovacie pravidla je výhodnejšie použiť zásuvné moduly knižnice PF_RING umožňujúce aplikovať filtrovacie pravidlo v rámci modulu jadra alebo rozpoznaním protokolu aplikačnej vrstvy. Použitie zásuvných modulov knižnice PF_RING umožňuje komplexnejší zber štatistik a informácie o obsahu komunikácie subjektu.

Ďalšou možnosťou zrýchlenia aplikácie je použitie PF_RING deskriptoru na odosielanie prijatých paketov. To je možné za podmienok, že rozhranie medzi serverom a klientov budem umožňovať prenosovú rýchlosť jedného gigabitu.

Toto prostredie je tiež možné simulovať prepojením vhodného hardvérového generátora spolu s aplikáciou dvomi rozhraniami pre okamžitú odozvu a meranie na strane generátora. Týmto vznikne kruhový test, na ktorom je možné porovnávať viaceré techniky zachytávania s okamžitým získavaním dát.

Urýchlením spracovaniana na strane server môžeme zabezpečiť už spomínaným aplikovaním pravidiel priamo na modul PR_RING. Týmto docielime výrazné zvýšenie rýchlosti odposluchávaného subjektu.

6 Testovanie

Testovacia architektúra východza z obrázku 5.5. Ten rozbrazuje softvérové aj hardvérove vybavenie komponent architektúry z ktorej sa odvíjal návrh aj implementácia.

Na testovacej architektúre bolo vykonávaných niekoľko sérii testov pre overenie schopností knižnice PF_RING prijímať pakety rýchlosťami približujúcimi sa rýchlosti prenosvej linky ako aj rýchlostí spracovávania jednotlivých paketov.

Pomocou zasielania signálou `SIGUSR1` a `SIGUSR2` sme vypisovali aktuálne nastavenie klienta a štatistiky knižnice PF_RING pfring_stats pre informácie o aktuálnom stave zachytených, zahodených a odposluchnutých paketov.

Využitím hardvérového generátora Spirent sme nastavili posielanie zhluku tísicich paketov s postupným nastavením veľkosti jednotiek MTU na hodnoty `128`, `512` a `1500` bajtov. Získané štastické výpisi vykazovali nulovú stratu paketov aj napriek využiťiu veľkosti MTU `128` bajtov a mechanizmu round robin pre generovanie paketov zmenou posledného bajtu IPv4 adresy. Rýchlosť bola nastavena na hodnotu jedného gigabitu za sekundu.

Získané výsledky možno odôvodniť faktom, že navrhnutá aplikácia využíva jednoduché filtrovacie pravidlo testované sekvenčným prechádzaním zoznamu s nastavením generovaných paketov tak aby bola dosiahnutá rýchlosť odosielajúcej linky pod sto megabitov za sekundu.

Získané výsledky reflektuje tabuľka 6.1. Kompletné výsledky testov sú k dispozícií na priloženom CD.

Výsledky testov v pre architektúru z obrázka 6.1.		
Konfigurácia: MTU: 128 Rýchlosť: 1 Gbit/s Round-robin: zmena posledného bajtu IPv4 adresy inkrementovaním hodnotu 0.0.0.1 4096 krát	**Konfigurácia:** MTU: 512 Rýchlosť: 1 Gbit/s Round-robin: zmena posledného bajtu IPv4 adresy inkrementovaním hodnotu 0.0.0.1 4096 krát	**Konfigurácia:** MTU: 1500 Rýchlosť: 1 Gbit/s Round-robin: zmena posledného bajtu IPv4 adresy inkrementovaním hodnotu 0.0.0.1 4096 krát
Výsledok: **Current time: Wed May 9 16:20:23 2012** **Intercept packets: 582906** **========== STATISTICS ==========** **Packets Received: 50840836** **Packets Dropped: 0** **All Packets: 50840836** **Dropped Packets: 0.0 %**	**Výsledok:** **Current time: Wed May 9 16:39:57 2012** **Intercept packets: 501310** **========== STATISTICS ==========** **Packets Received: 440058487** **Packets Dropped: 0** **All Packets: 440058487** **Dropped Packets: 0.0 %**	**Výsledok:** **Current time: Wed May 9 17:07:00 2012** **Intercept packets: 95162** **========== STATISTICS ==========** **Packets Received: 43222665** **Packets Dropped: 0** **All Packets: 43222665** **Dropped Packets: 0.0 %**

Tabuľka 6.1: Výsledky testov

7 Záver

Cieľom tejto bakalárskej práce bolo navrhnúť nástroj na spracovanie a záznam obsahu komunikácie. Tento cieľ bol splnený. Pred samotnou realizáciou bolo nutné sa oboznámiť s technikami pre zefektívnenie zachytávania vyžiadanej komunikácie subjektu. Tieto techniky sú zamerané na problémy vyskytujúce sa pri použití štandardných operačných systémov a bežne dostupného hardvéru, ktorými sú napríklad stratovosť paketov v súvislosti z rastúcou rýchlosťou prenosových liniek, či záťaži na hardvérové a softvérové vybavenie počítača z dôvodu náročného spracovania.

Nástroj pre zachytávanie obsahu komunikácie je navrhnutý s ohľadom na normy pre zákonné odposluchy úradom ETSI. Požiadavok je vydávaný právnou autoritou na základe podozrenie z páchania trestnej činnosti.

Využitím vhodnej techniky eliminujúcej problémy, ktoré nastávajú pri aktívnom zázname prebiehajúcej komunikácie možne zlepšiť dokazovacie metódy trestných činov. Touto technikou je PF_RING a s využitím sieťovej karty založenej na čipovej sade 82547L dosiahnutá rýchlosť jedného gigabitu bez stratovosti paketov. Testovacie techniky boli použíté na základe článku, kde sa prvý-krát vyskytol návrh schránky PF_RING [20]. Na základe vykonaných testov možno usúdiť, že použitá technika pracuje efektívne pri kopírovaní paketov zo sieťového rozhrania a tým zvyšuje možnosť aplikácie filtrovať pakety na základe sekvenčného prechádzania zoznamu bez veľkého opozdenia.
Ako prínos tejto práce vidíme v tom, že poskytuje odrazový mostík a zjednotenie informácii potrebný ch pre návrh a implementáciu nástrojov zaoberajúcich sa zachytávaním a spracovávaní paketov vo vysokých rýchlostiach. Porovnávaním viacerých techník spomínaných môže viesť k motivácii navrhovania ďalších testov a vývoji aplikácii so zmenou operačných systémov ako napríklad Windows, Free BSD či rôznych distribúcií Linuxu.

Mali sme možnosť oboznámiť sa bližšie so sieťovým podsystémom operačného systému Linux a jednotlivými technikami zachytávania paketov. Ďalším prínosom je získanie skúseností s programovaním a návrhom sieťových aplikácii v knižnici libpcap a PF_RING, ktorých požiadavok na efektivitu a celkovú výkonnosť testuje schopnosti programátora analyticky uvažovať a riešiť sporné body.

Testovaním aplikácie sme overili schopnosti knižnice PF_RING prezentujúce v literatúre [20] tým, že dokáže efektívne preniesť pakety z kruhovej pamäte do aplikácie bez zásadnej intervencie jadra a bez straty. Rozšírením pravidiel pre kontrolu paketov možno otestovať správanie sa PF_RINGu v prostredí bližšie k reálnemu fungovaniu. Spomínanou výhodou knižnice je úzka návaznosť na knižnicu libpcap, čo umožňuje existujúcim aplikáciam využiť rovnako potenciál kruhových vyrovnávacích pamäti ponukáných PF_RINGom. Tie je však nutné preložiť s modifikovanou verziou tejto knižnice zdrojový kód samotných aplikácii však nie je nutné meniť. Taktiež využitie modifikovaných ovládačov špecializovaných kariet spôsobuje nárast výkonu, no tie na druhú stranu ako býva zvykum u podobných riešení je ťažšie aktualizovať a tým zostať v kroku s vývojom hardvéru. Výhodou PF_RINGu je aj jednoduchá kompilácia modulu jadra a využitie jednoduchého API umožňujúceho vytvoriť kvalitné aplikácie na niekoľkých riadkoch kódu.

Do budúcnosti by sme mohli aplikovať získané poznatky konfrontovať s komerčnými systémami na odposluch komunikácie, využiť klasifikačné techniky pre aplikáciu filtračných pravidiel na zachytávané pakety čo by viedlo k ďalšej sérii testov a porovnávaní s vyhodnotením výsledkov. V tejto oblasti existuje veľké množstvo algoritmov, ktoré zefektívňujú proces zachytávania a ktorých vývoj ide neustále vpred v úzkej spolupráci s návrhármi hardvéru akcelerovaných sieťových kariet.

Táto práca by sa tak mohla stať vhodnou príručkou pre oboznámenie sa so základnými princípmi odposluchu a optimalizácii existujúcich riešení v tejto oblasti.

Literatúra

[1] Lawful Interception. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2012-02-19 [cit. 2012-05-10].
Dostupné z: http://en.wikipedia.org/wiki/Lawful_interception/

[2] ETSI TR 102 528. *Lawful Interception (LI)*: *Interception domain Architecture for IP networks*. 1.1.1. European Telecommunications Standards Institute, 2006.

[3] ETSI TS 101 671. *Lawful Interception (LI)*: *Handover interface for the lawful interception of telecommunications trafficIP networks*. 3.6.1. European Telecommunications Standards Institute, 2010.

[4] EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE. *Lawful Interception* [online]. 1996 [cit. 2012-05-10].
Dostupné z: http://www.etsi.org/website/technologies/lawfulinterception.aspx

[5] VARGHESE, George. *Network algorithmics*: *an interdisciplinary approach to designing fast networked devices*. Amsterdam: Elsevier, c2005, 465 s. ISBN 0-12-088477-1.

[6] BHUIYAN, Helali, MCGINLEY, Mark, LI, Tao, et al. TCP Implementation in Linux: A Brief Tutorial. In: [online]. [cit. 2012-05-12].
Dostupné z: http://www.ece.virginia.edu/cheetah/documents/papers/TCPlinux.pdf

[7] WU, Wenji, Matt CRAWFORD a Mark BOWDEN. The performance analysis of linux networking: Packet receiving. *Computer Communications*. 2007, roč. 30, č. 5, s. 1044-1057. ISSN 01403664.
Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0140366406004221

[8] LAWRENCE BERKELEY NATIONAL LABS. *Libpcap* [online]. Network Research Group, 2010 [cit. 2012-05-12].
Dostupné z: http://www.tcpdump.org/

[9] LAWRENCE BERKELEY NATIONAL LABS. *Tcpdump* [online]. Network Research Group, 2010 [cit. 2012-05-12].
Dostupné z: http://www.tcpdump.org/

[10] QUADEER, Mohamed Abdul, ZAHID, Mohamed, IQBAL, Arshad, et al. "Network Traffic Analysis and Intrusion Detection Using Packet Sniffer," *Communication Software and Networks, 2010. ICCSN '10. Second International Conference on*. 2010, s.313-317, 26-28.
Dostupné z: http://ieeexplore.ieee.org/stamp/
stamp.jsp?tp=&arnumber=5437681&isnumber=5437597

[11] Link Layer. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2012-4-30 [cit. 2012-05-12].
Dostupné z: http://en.wikipedia.org/wiki/Data_link_layer

[12] OSI model. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2012-5-11 [cit. 2012-05-12].
Dostupné z: http://en.wikipedia.org/wiki/OSI_model

[13] Ethernet frame. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2012-4-2 [cit. 2012-05-12].
Dostupné z: http://en.wikipedia.org/wiki/Ethernet_header

[14] Conventional PCI. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, Poslední modifikace 2012-4-2 [cit. 2012-04-16].
Dostupné z: http://en.wikipedia.org/wiki/PCI_bus

[15] HADI SALIM, Jamal, Robert OLSSON a Alexey KUZNETSOV. Beyond softnet: NAPI. In: *Proceedings of the 5th Annual Linux Showcase and Conference*. Oakland, CA, USA, 2001, s. 165-172.
Dostupné z: http://static.usenix.org/publications/library/proceedings/als01/full_papers/jamal/jamal.pdf

[16] Flow control: Hardware flow control. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2012-5-4 [cit. 2012-05-13].
Dostupné z: http://en.wikipedia.org/wiki/Flow_control

[17] SALAH, K. a A. QAHTAN. Experimental performance evaluation of a hybrid packet reception scheme for Linux networking subsystem. *Innovations in Information Technology, 2008. IIT 2008. International Conference on* [online]. 2008, s. 84-88 [cit. 2012-05-13].
Dostupné z: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4781671&isnumber=4781627

[18] TIANHUA, Liu, HONGFENG, Zhu, GUIRA, Chang, et al. Research and Implementation of Zero-Copy Technology in Linux. *Sarnoff Symposium, 2006 IEEE* , 2006, s. 1-4 [cit. 2012-05-13].
Dostupné z: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4534808&isnumber=4534705

[19] XIAOCHEN Xu a Li ZHONGWEN. High-Speed Packet Capture Mechanism Based on Zero-Copy in Linux. *Biomedical Engineering and Informatics, 2009. BMEI '09. 2nd International Conference on.* 2009, s. 1-5.
Dostupné z: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5305585&isnumber=5301644

[20] DERI, Luca. Improving Passive Packet Capture: Beyond Device Polling. [online]. 2003 [cit. 2012-05-13].
Dostupné z: http://luca.ntop.org/Ring.pdf

[21] NTOP. *PF_RING User Guide*. 5.3.1. 2012. [cit. 2012-05-14].
Dostupné z: http://www.ntop.org/pfring_userguide.pdf.

[22] WWW stránka: Ostinato - Packet/Traffic Generator and Analyzer. [online]. [cit. 2012-05-14].
Dostupné z: http://code.google.com/p/ostinato/

[23] WWW stránka: Intel® 82574L Gigabit Ethernet Controller. [online]. [cit. 2012-05-14].
Dostupné z: http://ark.intel.com/products/32209/Intel-82574L-Gigabit-Ethernet-Controller

[24] RFC 959 File Transfer Protocol. [online]. Vytvorené v Októbri 1985 [cit. 2012-05-14]
Dostupné z: http://www.ietf.org/rfc/rfc959.txt

Zoznam príloh

Obsah DVD

\bin	binárne súbory
\text	text prace a manuál k programu
\src	zdrojové texty programu
\test	výsledky testov z kapitoly 6 a množina použitých testovacích pravidiel